Malware Detection at Runtime for Resource-Constrained Mobile Devices

Data-Driven Approach

Doctoral Dissertation submitted to the
Faculty of Informatics of the Università della Svizzera Italiana
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Jelena Milošević

under the supervision of
Prof. Miroslaw Malek

December 2017

Dissertation Committee

Prof. Marc Langheinrich	Università della Svizzera italiana, Switzerland
Prof. Stefan Wolf	Università della Svizzera italiana, Switzerland
Prof. Alexander Romanovsky	Newcastle University, United Kingdom
Prof. Marco Vieira	University of Coimbra, Portugal

Dissertation accepted on 13th December 2017

Research Advisor	PhD Program Directors
Prof. Miroslaw Malek	***Prof. Walter Binder, Prof. Michael Bronstein***

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Jelena Milošević
Lugano, 13th December 2017

"We are at the very beginning of time for the human race. It is not unreasonable that we grapple with problems. But there are tens of thousands of years in the future. Our responsibility is to do what we can, learn what we can, improve the solutions, and pass them on."

Richard Feynman (1918-1988)

Abstract

The number of smart and connected mobile devices is increasing, bringing enormous possibilities to users in various domains and transforming everything that we get in touch with into smart. Thus, we have smart watches, smart phones, smart homes, and finally even smart cities. Increased smartness of mobile devices means that they contain more valuable information about their users, more decision making capabilities, and more control over sometimes even life-critical systems. Although, on one side, all of these are necessary in order to enable mobile devices maintain their main purpose to help and support people, on the other, it opens new vulnerabilities. Namely, with increased number and volume of smart devices, also the interest of attackers to abuse them is rising, making their security one of the main challenges. The main mean that the attackers use in order to abuse mobile devices is malicious software, shortly called *malware*.

One way to protect against malware is by using static analysis, that investigates the nature of software by analyzing its static features. However, this technique detects well only known malware and it is prone to obfuscation, which means that it is relatively easy to create a new malicious sample that would be able to pass the radar. Thus, alone, is not powerful enough to protect the users against increasing malicious attacks. The other way to cope with malware is through dynamic analysis, where the nature of the software is decided based on its behavior during its execution on a device. This is a promising solution, because while the code of the software can be easily changed to appear as new, the same cannot be done with ease with its behavior when being executed. However, in order to achieve high accuracy dynamic analysis usually requires computational resources that are beyond suitable for battery-operated mobile devices. This is further complicated if, in addition to detecting the presence of malware, we also want to understand which type of malware it is, in order to trigger suitable countermeasures. Finally, the decisions on potential infections have to happen early enough, to guarantee minimal exposure to the attacks. Fulfilling these requirements in a mobile, battery-operated environments is a challenging task, for which, to the best of our knowledge, a suitable solution is not yet proposed.

In this thesis, we pave the way towards such a solution by proposing a dynamic malware detection system that is able to early detect malware that appears at runtime and that provides useful information to discriminate between diverse types of malware while taking into account limited resources of mobile devices. On a mobile device we monitor a set of the representative features for presence of malware and based on them we trigger an alarm if software infection is observed. When this happens, we analyze a set of previously stored information relevant for malware classification, in order to understand what type of malware is being executed. In order to make the detection efficient and suitable for resource-constrained environments of mobile devices, we minimize the set of observed system parameters to only the most informative ones for both detection and classification. Additionally, since sampling period of monitoring infrastructure is directly connected to the power consumption, we take it into account as an important parameter of the development of the detection system. In order to make detection effective, we use dynamic features related to memory, CPU, system calls and network as they reflect well the behavior of a system.

Our experiments show that the monitoring with a sampling period of eight seconds gives a good trade-off between detection accuracy, detection time and consumed power. Using it and by monitoring a set of only seven dynamic features (six related to the behavior of memory and one of CPU), we are able to provide a detection solution that satisfies the initial requirements and to detect malware at runtime with F-measure of 0.85, within 85.52 seconds of its execution, and with consumed average power of 20mW. Apart from observed features containing enough information to discriminate between malicious and benign applications, our results show that they can also be used to discriminate between diverse behavior of malware, reflected in different malware families. Using small number of features we are able to identify the presence of the malicious records from the considered family with precision of up to 99.8%. In addition to the standalone use of the proposed detection solution, we have also used it in a hybrid scenario where the applications were first analyzed by a static method, and it was able to detect correctly all the malware previously undetected by static analysis with false positive rate of 3.81% and average detection time of 44.72s.

The method, we have designed, tested and validated, has been applied on a smartphone running on Android Operating System. However, since in the design of this method efficient usage of available computational resources was one of our main criteria, we are confident that the method as such can be applied also on the other battery-operated mobile devices of Internet of Things, in order to provide an effective and efficient system able to counter the ever-increasing and ever-evolving number and a variety of malicious attacks.

Acknowledgements

Throughout the course of my PhD journey I learned a lot of new things, got engaged in numerous discussions, traveled to new places and grew up as a person in various aspects. Most importantly, I met many new people and a lot of them provided me their help, advice and support, making it an amazing adventure. With each person I share a story, an experience from which I learned something new and useful. Since saying all these stories and mentioning all the people is not quite possible, I will acknowledge here only one part of them, the ones that were the most influential for my studies, but I am very thankful also to all the others I met along the way.

First of all, I would like to thank to my research advisor, Professor Miroslaw Malek, for all his support and advice over the years. Since the beginning of my studies he was more than supportive. We had numerous discussions on different research ideas and possibilities, that helped me to improve critical way of thinking, to better shape-up the ideas and to present and write them in more clear and precise manner. Additionally, he always supported my ambition to learn more and gain new experience, and thanks to this I attended different summer schools and courses, that helped me to further increase my knowledge, to meet other peers and to discuss my research with them. Together with my advisor, I went through various situations: from painful rejections to comments like 'the paper is beautifully written', from kebab lunches in mensa to exquisite pastas in Anema&Core, from participation in ALaRI bike racing to Christmas bowling, etc. Now, when I am preparing to embark on a new, PostDoc journey, I am very grateful for all the guidance I had, and I am ready to apply what I learned until now and to support younger students with the same level of enthusiasm and attention that I received from him.

When it comes to discussion on research ideas, help in going through painful reviews, and advice on how to improve quality of writing, one other person was of enormous help to me. That is my colleague Dr. Alberto Ferrante. I started working with Alberto even before starting my PhD, and this helped me to understand better many things related to the field of information and system security,

Linux Operating System and design of embedded devices, and to later define the direction of my PhD more easily. We worked together on four different projects and up to date, co-authored together 13 publications, but, hopefully, even more will come in the future (not to stay with this unlucky number). Through more than five years of working together, we always had only one problem: to convince Alberto to join us in mensa for lunch. I hope this will happen at least once, maybe after my defense, but let's see.

Another colleague, whose advice were of utmost use to me, is Dr. Francesco Regazzoni. Francesco's comments on the research ideas presented to him or suggestions on the papers in progress, were always reflecting the potential comments of the harshest reviewer one could get and the discussions were usually starting with 'Do you think this paper is the candidate for the best paper award?'. In this way, he always motivated me to work harder and to aim at the highest goals. Although I will miss him when I leave Lugano, knowing how frequently he travels and visits different places, I believe I will be seeing him often also in Vienna, and I feel that also in the future I will have many similar discussions with him, if not in person, then via the oldest version of Skype available.

In addition to Alberto and Francesco, two other colleagues, especially at the beginning of my studies, were of huge help to me: Dr. Onur Derin and Dr. Andreas Dittrich. At the point when I met Onur, I was at the beginning of my internship at ALaRI, and he was already fourth year PhD student. His experience helped me to go much faster through initial confusion of changing place to live, dealing with administrative things in an unknown language, and getting familiar with new place. Something similar, but more related to the exact research activities, was also the case with Andreas, with whom I had pleasure to work with during the first year of my PhD, when it was the most important for me to frequently discuss the details related to the malware detection methodology that we propose in this thesis. From Andreas I learned that it is very important to structure well from the early beginning the collected references, related work, and most importantly, to create a Redmine (project management software that we use at ALaRI) issue for each problem that I would encounter, in order to easily keep track of them and to make the collaboration easier. Also, thanks to Andreas I started using Python, which turned out to be a useful tool for various tasks I had later on. Unfortunately, my progress in using Python was not followed by progress in learning German, but there is still time to improve in that domain too, and I am confident that one day we will be drinking together weiss-bier either in Berlin or in Vienna and at the same time converse fluently in German.

In addition to the aforementioned people from ALaRI, that were not only my colleagues for previous five years but also became my friends along the way, I

am also thankful to Professor Maria Giovanna Sami, Professor Cesare Alippi, and Umberto Bondi, with whom I always had very useful discussions from which I was able to learn something new and important for the next step of my studies.

I also wish to thank Professors Marc Langheinrich and Stefan Wolf from Faculty of Informatics, Professor Alexander Romanovsky from Newcastle University, and Professor Marco Vieira from University of Coimbra for time and effort spent in reading my dissertation proposal and for useful comments that they provided me. These comments helped me to shape this thesis better, motivated me to continue with my envisioned research tasks and guided me in the direction of the PhD completion.

In addition to the research activities related to the PhD and participation in the projects of the institute, I was also involved in the teaching activities in the Faculty of Informatics. Related to this I would like to thank again Professor Marc Langheinrich, with whom I was a TA in the Information Security course, for his support and help during learning how to approach that, at the time, new task for me. I had the opportunity to participate in the development of the course structure from its early beginning of planning the content of the lessons up to grading final exams of students, and in this way could understand how does the whole process function and be more ready to apply it in the future, if needed. Also, in various discussions with Professor Langheinrich and while attending his lectures, I learned a lot about different aspects of Information Security, particularly cryptography and network security, for which I am very thankful.

Being a PhD student, the most of the time I spent with other PhD students engaged in various research and non-research related activities. I first wish to thank ALaRIan PhD students who started their studies before me: Rami Baddour, Igor Kaitović, and Katarina Balać, who shared their experience with me once I joined the Institute, and then to Felipe Valencia (Felipe, sorry for not writing all your five names here, the space is limited) and Daniele Zambon, who joined the Institute after me, and brought additional enthusiasm and freshness into the group. The casual moments we spent drinking coffees during the day, or discussing feature selection methods and Weka tool, or going to a gym and aquarobics together or going for lunch in mensa, eventually summed up in a lot of time spent together, that I consider now as one of the most precious aspects of the PhD time spent in Lugano. Being drawn into a hectic PhD life, first years were passing quickly for me without noticing it and with constant feeling that I am still at the beginning of my studies. However, the departure of three good friends of mine: Dr. Elena Khramtcova, Dr. Artiom Kovnatsky and Dr. Dmitry Anisimov, who finished their PhD at USI, and with whom I shared much of my time during the years, made me understand that the phase I was in at that point was already the fin-

ishing phase rather than the beginning one, that helped me to be more focused on the envisioned tasks and to carry them out faster. This was particularly the case with Dima, after whose departure going to the Irish pub was not such an adventure anymore, neither was so interesting to drink cappuccino from a coffee machine on the bench in front of the university. Having Dima, Dasha and Batman in Lugano was like having close family that one can always rely on. Having such friends I consider one of the biggest success in life for which I am very thankful, and I am confident that in the future we will again have opportunity to spend time together, if not by sharing beer and cappuccino, then by sharing wine and baguette (I do not dare to add here 'and by speaking in French').

Living in the Italian part of Switzerland had many advantages, one of which was good quality of coffee, to which I got used and that was very important for proper progress in PhD. Thanks to this fact, I also meet with Dr. Francesco Mercaldo (then from the University of Sannio and now from CNR in Pisa), since we were at the same time complaining about the quality of coffee served at the conference. Apart from the same taste for good cappuccino before the keynotes and wish to have short espresso after lunch, it turned out that also our research areas are overlapping. Already during the conference we discussed different mutual research directions, and since then we are collaborating on different projects related to malware detection. A lot of Francesco's work was in the area of static analysis of mobile malware and for the evaluation of the hybrid method that we propose in this thesis, we actually use the static classifier proposed by him. Thanks to Francesco, I also met with Professor Eric Medvet, from University of Trieste, with whom I had a lot of fruitful discussions related to the machine learning methods and their possible application to the malware detection, and with whom I worked a lot when we were developing the detection system being able to identify groups of malicious executions.

During the course of my studies, I did two internships: first one at IBM Cyber Security Center of Excellence in Beer Sheva, Israel and the second one in Movidius an Intel Research and Development Company in Dublin, Ireland. At IBM I extended my knowledge in anomaly detection methods, for which I am very thankful to Dr. Eitan Menahem, with whom I was collaborating during the internship. Also, many thanks to Dr. Yaron Wolfsthal, the Associate Director of the center, and to Professor Shlomi Dolev from Ben-Gurion University of the Negev, with whom I had many fruitful discussions before and during the internship. Equally rewarding experience was also the time spent at Intel, for which I am thankful to the colleagues with whom I worked there: Andrew Forembski, Dr. Dexmont Peña and to the Director of Machine Vision Technology at Intel, Dr. David Moloney. Time spent at Intel helped me to understand the enormous

potential of deep learning and to learn how to apply it in the domain of the embedded devices. I hope that our mutual paper will be published soon, so that we can discuss with the research community all the aspects of deep learning and its usage in embedded systems that matter.

During my studies at Faculty of Technical Science in Novi Sad, Serbia, I had opportunity to meet many great professors, from whom I aquired knowledge that later helped me during my PhD. The ones that influenced my later studies the most were my master thesis advisor Professor Dragana Bajić (Full Professor at Faculty of Technical Science), and my master thesis co-advisor Dr. Čedomir Stefanović (now Associate Professor at the Aalborg University, Denmark). Thanks to them, after my master thesis, I published my first research paper, that later got awarded as the best student paper, which further motivated me to continue with research. Leaving Serbia and going abroad was a difficult step for me since I had amazing time, amazing friends, and my whole family there. During the years of PhD, I was frequently too busy to keep in touch with all of them, but each time I was there for a visit I was facing more than understanding from their side, and our friendship stayed as strong as before. For this reason I am very thankful first to Dr. Nebojša Božanić, my good friend already from the high school (I still remember how it was a lot of fun preparing the high school qualification exam with him), then to Nemanja Živković, Ivana Krstić and Stanko Knežević, who were my colleagues during bachelor studies and since then became valuable friends with whom I travelled to many different places and spent a lot of time together. Also, many thanks to Jovana Lugonja, who was my project companion during master studies. We learned a lot of things together, but more than that, we became friends that can rely on each other.

After the completion of my master studies, I had a pleasure to work for almost one year in Zesium mobile and RT-RK Institute, under supervision of Dr. Željko Lukač, where I learned a lot of new things related to the software development for embedded systems, optimization of code for audio and video codecs, and writing and understanding Assembly, for which I am very thankful, and which were extremely useful in my later work. But once again, I am most thankful for spending my time with great people there, with Darko Lauš, Dragan Mrđan, Mirjana Vulin and Bojan Živković, that I am still in contact with and that I am always happy to meet again and discuss again and again MIPS instruction set and optimization of matrices multiplication. Additionally, while working in RT-RK, I met two amazing friends, now wife and husband, to whom I was the best woman at the wedding: Uglješa Dragišić and Mirjana Đurić Dragišić. Thanks a lot for your support throughout the years of studying and I hope we will recover this time spent living in separate countries. I also would like to thank Dr. Srećko Stević,

whose company and support for multiple years was of the great importance to me and helped me to overcome some of the most difficult obstacles before and during the PhD path. My gratitude goes also to Dr. Đorđe Marić, who always motivated me to pursue the academic path and who was always supportive during the years of my PhD.

Although journals are certainly the most influential type of scientific publications, in my case it was one poster that made a huge difference. Namely, while presenting a poster I met Dr. Yanick Fratantonio, from University of California, Santa Barbara, also a PhD student then. Since the early beginning we had a lot of common mutual interests: mobile security, digital privacy, and above all passion for graph isomorphism. Getting to know Yanick was one of the best things that has happened during my studies, because from his example I learned how to incorporate joy as a part of all daily activities, instead of considering it as an exclusive phenomena left only for vacation period. This was particularly useful because since then I did not have to go to a proper vacation anymore, which was money and time saver, and which always came in handy in the PhD students way of living. Up to now, although we have conducted several experiments and collected some initial results already, we still do not have a mutual publication, and I hope that it will change in the future and that the graphs isomorphism application in Android malware detection domain will happen.

The ones that unconditionally supported me since the beginning of my educations were my family members. Talking to my brother and his family was always bringing positive and happy component in my day, reminding me of what are the most important values in life and how other things that might seem big from a short distance, from a wider perspective might be almost insignificant or at least not as significant as they seem at the first glance. Many thanks also to my parents, although saying just thanks to them sounds so little and insufficient for all that they did for me in the previous time, for all the time spent in worrying if I arrived safely to different destinations, or if my paper got accepted or rejected, or if I am going to pass my prospectus/proposal and so on. My father was always supporting me in pursuing further education, in further improving all the aspects of my life, in exploring new places and trying new experiences. I recall him frequently telling me the following words of Sir Winston Churchill: 'Who owns the information, owns the world.' Finally, my mother was always contributing to improvement of quality of what I am doing, the way I was doing it, and my attitude towards it, reminding me frequently of common Serbian proverbs, that translated from Serbian can be said in the following way: 'When you are working, work,' 'What you can do today, do not leave for tomorrow,' and finally 'Three times measure, one time cut.' A million thanks for everything.

Contents

Contents **xiii**

List of Figures **xvii**

List of Tables **xix**

1 Introduction **1**
1.1 Main Characteristics 4
1.2 Main Contributions 4
1.3 Thesis Outline 5

2 State of the Art: Mobile Malware Threats and Solutions **9**
2.1 Mobile Threats 10
2.2 Existing Malware Detection Solutions 14
2.2.1 Static Detection 14
2.2.2 Dynamic Detection 18

3 Proposed Methodology for Malware Detection at Runtime **25**
3.1 Overview of Proposed Methodology 26
3.2 Selected Features Monitoring 28
3.3 Malware Detection 29
3.3.1 Malicious Records Detection 30
3.3.2 Malicious Applications Detection 31
3.3.3 Detection of Malicious Sub-traces 32
3.3.4 Optimization of the detection solution by changing sampling period of the monitoring infrastructure 34
3.3.5 Application-specific usage of the proposed methodology . 35
3.3.6 Computational overhead 36
3.4 Malware Classification 36
3.5 Hybrid Method 38

3.5.1 Static Analysis . . . 39
3.5.2 Dynamic Analysis . . . 41
3.6 Key Properties of the Proposed Approach . . . 42

4 Setup for the Experiments 45
4.1 Datasets . . . 45
4.1.1 Benign applications collection . . . 46
4.1.2 Malicious applications collection . . . 46
4.1.3 Dataset used in malware detection task . . . 48
4.1.4 Dataset used in malware classification task . . . 48
4.1.5 Dataset used in the evaluation of the hybrid method . . . 49
4.2 Database creation . . . 50
4.3 Collected features . . . 52
4.3.1 Memory and CPU related features . . . 52
4.3.2 Network behavior related features . . . 53
4.3.3 Observed system calls and features related to the statistics of system calls . . . 53
4.4 Features Selection . . . 54
4.5 Power Consumption Measurement . . . 56
4.6 Development Environment . . . 57

5 Experiments and Results 59
5.1 Identification of the most indicative features for efficient and effective malware detection and malware classification . . . 61
5.2 Malware Detection . . . 64
5.2.1 Identification of the most indicative features for malware detection . . . 64
5.2.2 Runtime Detection of Malicious Records . . . 66
5.2.3 Detection of Malicious Sub-traces . . . 69
5.2.4 Malicious applications detection . . . 72
5.2.5 Optimization of Sampling Period of Monitoring Infrastructure and Power Consumption . . . 76
5.3 Malware Classification . . . 83
5.4 Performance Evaluation of a Hybrid Method . . . 86
5.4.1 Static Detection Results . . . 87
5.4.2 Dynamic Detection . . . 87
5.4.3 Hybrid Detection . . . 90
5.5 Limitations of the Proposed Approach . . . 90
5.6 Comparison of the Obtained Results with State-of-the-art Methods 92

6 Conclusions and Future Work **95**
6.1 Conclusions 96
6.2 Future Work 97
6.2.1 Are static or dynamic features more resistant against unseen malware families? 97
6.2.2 Is deep learning a good candidate for malware classification task, and if yes, can we have timely decisions using it? 98
6.2.3 Would we benefit from a dedicated hardware accelerators for on-device malware detection? 99
6.2.4 What are the performance of our detection methods if applied in the other elements of the IoT infrastructure? . . . 99
6.2.5 Using this methodology can we detect cyber attacks in smart grid? 99

Bibliography **101**

List of Figures

3.1 Malware detection and malware classification system. 27
3.2 Offline development of the malware detection mechanism. 28
3.3 Offline development of the malware classification mechanism. . . 28
3.4 The proposed detection system's offline blocks and its runtime components. 30
3.5 An example of the sliding window algorithm applied to a sequence of classified execution records; window length, threshold, and number of checks are set to 4, 70%, and 2, respectively. 32
3.6 High-level workflow of the proposed hybrid approach. The static analysis consists of a classification using features obtained by the executable of the application under analysis, while the dynamic one by a feature set obtained when the application is running. . . 39

5.1 Number of occurrences of memory and CPU usage related features among the top 5 most indicative ones 62
5.2 Number of occurrences of memory and CPU usage related features among the top 15 most indicative ones 63
5.3 Average ranking of the features that appear among the top 5 the most indicative ones. Blue color depicts the average rank of the observed features, while red color depicts its number of occurrences. 63
5.4 Average ranking of the features that appear among the top 15 the most indicative ones. Blue color depicts the average rank of the observed features, while red color depicts its number of occurrences. 64
5.5 Optimized J48 Decision Tree; malicious and benign traces are labelled with *0* and *1*, respectively. 68
5.6 F-measure obtained with different sampling periods and the sliding window algorithm parameters that maximize F-measure. . . . 77
5.7 F-measure, power, and detection time obtained for all configurations (sliding window algorithm parameters and sampling periods). 79

5.8 F-measure, power, and detection time obtained for configurations (sliding window algorithm parameters and sampling periods) that have acceptable performance. 79
5.9 F-measure vs. power for all the configurations that have acceptable performance. 80
5.10 F-measure vs. detection time for all the configurations that have acceptable performance. 81
5.11 Power versus time for all the configurations that have acceptable performance. 81
5.12 Metric values of all the configurations that have acceptable performance. 83

List of Tables

4.1 Considered categories for analyzed benign applications taken from GooglePlay. 47
4.2 Malware detection dataset description 48
4.3 Malware Classification dataset description 49
4.4 Hybrid approach dataset description 50
4.5 List of all the considered features; totals are related only to single applications; unless differently specified, all numbers are related to the considered sample rate of 2s. 54

5.1 Top 15 features ranked by different feature selection algorithms. . 65
5.2 Performance of the classifiers when different number of features is considered. 67
5.3 Brief description of the most indicative features Google Inc. [2015]. 69
5.4 Sub-trace accuracy. 71
5.5 Algorithm parameters used in the two phases of the exploration. . 73
5.6 Best malicious applications detection results obtained in the coarse-grain exploration of detection algorithm's parameters. 74
5.7 Best malicious applications detection results obtained in the fine-grain exploration of detection algorithm's parameters. 75
5.8 Best malicious applications detection results attained on the validation dataset with the parameters obtained from fine-grain exploration. 75
5.9 Best results obtained in the algorithm parameters exploration when different sampling periods are considered. 77
5.10 Validation with different sampling periods. 78
5.11 Best configurations for the observed optimization criteria. 82
5.12 Representative features of the observed malicious Trojan families 85
5.13 Obtained malware classification results 86

5.14 Classification results for malicious and benign applications when features extracted by static analysis are considered along with the J48, NB and LR classifiers. 87
5.15 Classification results for malware and benign applications when features extracted by dynamic analysis are considered along with the J48, NB and LR classifiers. 88
5.16 Best results obtained in coarse-grain exploration. 89
5.17 Best results obtained with respect to the observed metrics. 89
5.18 Results obtained in the testing phase with the best parameters obtained in the exploration phase. Detection time is measured from the first record identified as malicious. 89

Chapter 1

Introduction

Mobile devices have conquered all aspects of personal and professional life: smart watches are used to track our daily activities and the quality of our sleep; mobile applications are used to perform financial transactions; vehicles contain a significant amount of interconnected computational elements that are used to control their functionality, ranging from fuel injection to infotainment. Critical infrastructures, such as smart grids and public transport, also make use, at different levels, of mobile and battery-operated devices. With the widespread adoption of Internet of Things enabled devices, the presence of mobile computational elements is growing even faster and, the focus of Internet security is shifting from the desktop and the data centers to mobile and IoT devices. It is already estimated that the number of connected devices will continue to grow both in volume and variety, and, that by 2020 it may reach 20 billion according to Gartner [2017].

Widespread adoption of smart mobile devices and their increased usage for personal and business purposes, attracted the attention of attackers, mainly criminals, and increased their interest in abusing these devices in order to gain profit, collect private and sensitive data, or disrupt users. This reflects in an increase of malware, by which we consider any malicious software that gains access to a device for the purpose of stealing data, damaging the device, or annoying the user, as reported by Felt, Finifter, Chin, Hanna and Wagner [2011]. Malware is currently one of the most relevant security problems of mobile devices, since number of encountered malicious samples is constantly on the rise, and, according to *McAfee Labs Threats Report* [2017], total mobile malware grew by 79% in the past four quarters to reach 16.7 million samples. Additionally, as stated in 2016 Trend Micro Security Predictions Micro [2015], 3 out of 4 applications used in China are believed to contain malware.

The increase of malware, calls for an increase in the effectiveness of malware detection systems. Although mobile malware detection systems are used in environments with limited computational resources, in order to be adopted in practice and to help users to defend against malicious infections, they have to fulfill a set of requirements. First requirement is high detection accuracy, without producing too many false positives and without disturbing regular usage of the device. Then, an infection with malware should be detected as early as possible in its execution, in order to minimize potential damage. Additionally, the overhead on battery and computational resources consumption due to detection system should be negligible. Finally, the detection system should provide information on which of the various types of malware are executing, in order to enable better understanding of severity of the attack and to propose possible countermeasures. However, with the increased number of malware samples and a currently existing plethora of malicious behaviors, these are challenging tasks, that are additionally complicated with the fact that constrained resources and battery-operated nature of mobile devices significantly limits their ability to run complex malware detection systems, as stated in Zhou and Jiang [2012], so in most practical scenarios the trade-offs between the mentioned requirements have to be considered.

Two existing ways to detect malware are static and dynamic analysis. Static analysis is based on the offline investigation of the applications by means of static features, that can be analyzed without the execution of a program. Although it is efficient, it cannot cope with increased number of malware samples and their variations. It is prone to obfuscation, and alone may no longer be sufficient to identify malware, as reported in Moser et al. [2007]. The other method, dynamic analysis, is based on observing dynamic features, that change during the execution of the program. Since it is focused on observing systems behavior at runtime, in such a way malware cannot easily hide. Although in the most of the proposed detection solutions these two methods are considered separately, according to McAfeeLabs [2016], *"Security software should include dynamic analysis to flag rogue actions regardless of initial binary inspection because static scanning goes only so far."* However, dynamic analysis and its usage at runtime require algorithms that are frequently too complex for battery-operated mobile environments. Additionally, most of the currently proposed dynamic detection methods, apart from being computationally complex, provide the ability to detect only complete applications as malicious or benign, without providing any insights on which parts of the application executions are actually malicious, neither to which group of malware they belong to. Understanding which parts of the executions are malicious is useful for better training of malware detection systems. Addi-

tionally, having insights about the behavior of the parts of the applications, helps a detection system to build an overall image of trustworthiness of the executing software and, if needed react very early if the system is potentially compromised and the detection time is of the utmost importance, or react when a certain confidence is built, in scenarios where high detection accuracy and low false positive rates have priority. Identification of what type of malware could be running on a mobile device is important, in order to understand better the goal of the attack and how it aims to infect the device, to put in place suitable countermeasures, and potentially minimize the damage the attack can cause. However, having in mind the constrained computational environment of mobile devices, runtime detection of malware and, particularly, identification of type of malware being executed (malware classification) are challenging tasks, and most of the existing works fail to provide optimized solutions that balance well between high malware detection accuracy, accurate malware classification, early detection of malware and low resource consumption.

Most malware infections happen when users download and start applications that contain hidden malicious payloads. This can happen either by chance, when users are tricked to download some content via fishing or social engineering methods, or intentionally, when users try to avoid official version of applications that cost money and instead download their free versions available on third markets. Additionally, malicious content can be downloaded even after the application installation during its update or can be hidden in a legitimate application and activated only in certain conditions.

Malware, as a widespread threat, is present in a variety of mobile devices and almost all mobile operating systems. However, the ones that are particularly affected are mobile devices running on Android Operating System (Android OS). This is the case due to the widespread usage of Android OS and its dominance on the mobile market. Having in mind widespread usage of Android OS not only in mobile phones, but also smart TVs, smart watches, and as a potential candidate for future IoT standard operating system, protection of Android devices from malware is a relevant security problem that has to be addressed. In addition, detailed analysis of Android malware, performed by Zhou and Jiang [2012], revealed that the dynamic loading ability of both native code and Android-specific Dalvik code are being actively abused by existing malware. Having in mind all the considered scenarios of malicious payload, the most suitable way to detect it is dynamically at runtime, during applications executions on the device.

1.1 Main Characteristics

In this thesis we focus on *design, test and validation of a malware detection system suitable for resource-constrained mobile devices*. The system has following characteristics:

a) it is suitable to be used at runtime on resource-constrained devices

b) it maximizes detection performance

c) it minimizes detection time

d) it is able to identify at runtime parts of malicious applications

e) it is able to identify at runtime complete malicious applications

f) apart from discriminating between benign and malicious behavior, it is able to discriminate also between diverse behavior of malware

g) it can be used as a standalone method or in addition to static methods, as a part of a hybrid solution to malware detection problem

h) it can be demonstrated on high-impact operating system such as Android OS.

1.2 Main Contributions

The main assumptions that we consider are that the observed environment is stationary and that the malicious payload is triggered in the observed environment Towards design of the envisioned detection system, under these assumptions, we tackled and contributed to the following research problems:

- Design of a methodology that performs malware detection on device and, only when a potential attack is detected, sends information about it to a remote server to perform malware classification, taking into account a trade-off between low resource consumption and suitability for battery-operated devices on one side, and on the other, the accurate identification of malware at runtime and its accurate classification
- Design of a methodology for efficient and effective detection of mobile malware at runtime for resource-constrained devices
- Creation of a comprehensive database consisting of execution records of both malicious and benign applications

- Identification of the set of the most indicative dynamic features to be observed in order to discriminate between malicious and benign behavior
- Identification of the set of the most indicative features for discrimination between different malicious behavior (malware classification)
- Design and development of detection algorithms suitable for resource- constrained environment and able to detect parts of malicious applications
- Design and development of detection algorithms suitable for resource- constrained environment and able to detect complete malicious applications at early stage of their executions
- Investigation of the importance of the sampling period of monitoring infrastructure (i.e., how often the behavior of the apps is monitored to detect malware) for efficient malware detection and evaluation of the system power consumption with respect to it
- Evaluation of the system detection performance with respect to detection accuracy, detection time and power consumption
- Description of functional dependencies between detection quality and detection time, detection quality and resource consumption, and detection time and resource consumption
- Evaluation of designed and developed detection methods in a hybrid scenario, where the observed applications passed static analysis first
- Demonstration of the application of detection methods to high-impact operating system such as Android OS.

1.3 Thesis Outline

With increased interest of attackers to abuse mobile devices, the number of attacks performed by malicious software is rising, so as the types and varieties of existing threats. In order to deal with this increase and to protect mobile devices, different mobile malware detection solutions are proposed. In Chapter 2, we discuss the existing mobile threats and proposed detection solutions. More in detail, in Section 2.1 we explain: rootkits, ransomware, bots, financial malware, logic bombs, viruses, worms and Trojan horses, that are the most present malware types in mobile devices. Currently, there are two ways to cope with the existing

malware: through static or dynamic analysis. We discuss the existing approaches based on these two methods in Sections 2.2.1 and 2.2.2, respectively.

In Chapter 3, we present the approach we propose in order to achieve an efficient and effective detection of malicious parts of applications, complete malicious applications and their intentions (malware families they belong to). Our main goal is to provide malware detection solution that can be suitable for runtime usage on resource-constrained mobile devices and that can detect malicious attacks in the early stage of their execution. To outline of the approach we introduce in order to reach this goal is given in Section 3.1. One of the most influential aspects of the proposed methodology is identification of the features that are indicative of the presence of mobile malware. In Section 3.2 we describe the origin of features that we observe in our system, the reasoning behind it and the motivation behind performing feature selection as a separate task. Then, in Section 3.3 we describe the proposed malware detection module and the approach we use in order to perform efficient detection of malicious records, detection of malicious applications and detection of groups of malicious records (sub-traces). In Section 3.4 we discuss the proposed malware classification module. The approach we propose and evaluate in this thesis is dynamic and can be used as a standalone malware detection system. However, it is also possible to use it as a part of a hybrid approach, in which, we would first use a static detector to perform offline investigation of the application's intentions, and then, only if the application is marked as benign, we would run it on a phone, observe its characteristics at runtime, and warn users in case it starts to perform malicious actions. In Section 3.5 we describe the characteristics of such a hybrid approach. Finally, in Section 3.6, we summarize the main properties of the malware detection approach we propose.

In Chapter 4, we describe the experimental setup we have implemented for the mobile devices running Android OS, the dataset that we used, and the features we collected. The experiments were performed using both benign and malicious applications. The collection of these applications is described in Section 4.1. By executing the collected applications in the Android OS environment and observing their influence on system parameters of devices we obtained a set of execution traces that we store in a database that is described in Section 4.2. The system parameters taken into account (memory, CPU, network usage and system calls) are listed in Section 4.3 and the methods used in order to perform their extraction and selection of the most indicative ones are given in Section 4.4. For the evaluation of the power consumption of the developed detection methods we used the setup described in Section 4.5.

In Chapter 5 we present the obtained results on detection of malicious ex-

ecution records, malicious applications, discrimination between different malware families and detection of malicious parts of the applications. The identification of important features for malware detection and malware classification tasks is first described in Section 5.1. Since the results obtained in the domain of malware detection are one of the main contributions of the thesis, we discuss them in Section 5.2, where we give detailed description of the selected indicative features, detection performance of performed malicious records detection, malicious sub-traces detection and of malicious applications detection. Together with the malware detection, we also propose a methodology for malware classification, and the results obtained for this task we discuss in Section 5.3 where we outline the features indicative for the presence of the observed malware families and the detection performance of selected classifiers when these features are used. When we apply our dynamic detection method on applications previously scanned with static detection, in form of a proposed hybrid scenario, we achieve high detection performance that we discuss in Section 5.4. The method we propose for the runtime detection of malware is based on a dynamic analysis and usage of machine-learning methods, which means that it inherits some of their disadvantages and that it has some limitations, that are discussed in Section 5.5. To the best knowledge, the detection approach we propose, design and validate is different in what it offers than all other state-of-the-art methods in various aspects. These aspects are discussed in Section 5.6 where the proposed method is compared to the existing works in the field of mobile malware detection.

Finally, in Chapter 6 we conclude the thesis, and describe the future work. Section 6.1 summarizes the obtained results and Section 6.2 outlines the steps envisioned for the future.

Chapter 2

State of the Art: Mobile Malware Threats and Solutions

Mobile systems are often network connected, and this connectivity can be exploited to attack them, as stated by Clark and Fu [2012]. A common way of performing security attacks on computers and different types of devices is to inject malicious software called *malware*.

As pointed out by Viega and Thompson [2012], malware has been found in all kinds of cyber-physical systems over the years. It has the goal of stealing sensitive information from users, taking control over the operating system, and damaging or even completely disabling the device. Malware can be used to directly attack devices or as an enabler for other attacks. Devices may get infected in different ways. Malicious software can be covertly installed as pointed out by Khan et al. [2012], and often with the collaboration of unaware users through the installation of applications. It has been shown in Percoco and Schulte [2012] that even applications downloaded from official websites may contain hidden malicious portions of software. Other ways to install malware on devices include the exploitation of operating system vulnerabilities (e.g., through the Internet connection, e-mails, videos or websites).

The rise of mobile, connected devices led to a plethora of mobile malware opportunities, exploiting known and creating new attack vectors, as stated in literature in Dunham [2008], in Delac et al. [2011], and in Guo et al. [2007]. For the second consecutive year, mobile devices are perceived as IT security's weakest link, according to CyberEdge Group [2015], and, as pointed out in *Internet Security Threat Report Volume 20* [2015], the threats, that were previously mostly concern of governments, financial institutions, and security vendors, are becoming more relevant in small enterprises and in personal lives. The same

report also states that the focus of Internet security is shifting from the desktop and the data center to the home and Internet of Things, the pocket, the purse, and, ultimately, devices and infrastructure of the Internet itself.

In the remaining part of this chapter we discuss the most relevant mobile threats up to date and we survey existing detection solutions. A deeper and broader analysis about issues and techniques for securing the Android platform may also be found in two extensive and recent surveys by Faruki et al. [2015] and by Tan et al. [2015].

2.1 Mobile Threats

The most severe threat that can affect mobile devices, as stated in Milosevic, Ferrante and Regazzoni [2015], is malware. It is being able to completely damage a device or enable further attacks on the device that can perform unwanted actions. Most present malware types in mobile devices are rootkits, ransomware, bots, financial malware, logic bombs, viruses, worms and Trojan horses (Trojans). More details about each of them are as follows:

- **Rootkits** are a type of malware that is able to access parts of software for which regularly it does not have privileges. The access to privileged area is usually enabled by performing an attack on the system, either by exploiting systems vulnerabilities or guessing users passwords. Once the attacker has the access to the root privileges of the system, the system is practically under full control of him or her and is prone to further manipulation. Due to this, rootkit detection is a challenging task. In the domain of Android malware, using root exploits is a common way to perform attack on the system. Zhou and Jiang [2012] performed detailed analysis of Android malware from this perspective and based on the obtained results stated that out of 1,260 analyzed malicious applications representative for Android malware landscape, 463 of them (36.7%) embed at least one root exploit and 378 samples came with more than one root exploit.

- **Ransomware** is malware that locks the content of the users device, and then asks the user to pay money, ransom, in order to enable normal usage. There are different ways to perform such attack on the system, starting from locking the screen of a device, or by using fake anti-virus software that, once installed on users device, would prompt the message that the device is under attack and ask for money in order to remove the discovered infection. More advanced ransomware encrypts the data stored on the

device and asks for money in order to provide the decryption key. In the last few years an increased number of ransomware attacks was recorded. More in detail, according to *McAfee Labs Threats Report* [2017] the number of total ransomware samples grew 59% in the past four quarters to reach 9.6 million samples. According to McAfee Labs [2016], 2016 will be remembered as "the year of ransomware," confirming the predictions of exponential growth of these kinds of attacks from previous years given by Infosec Institute [2016]. Recent events, such as the WannaCry ransomware attack of May 2017 in which over 200,000 computers in more than 150 countries were rendered unusable with ransom demands [1], demonstrated that these predictions might be beaten in 2017.

- **Bots** are self-propagating malware with the goal to infect host machine and later connect to a server, bot master, and follow the obtained orders from it. Botnet is a network consisting of many host devices infected with bots, being available to perform Denial of Service attacks, send spam messages or simply enable further infections on host devices. Additionally, bots collect information from host devices and send it to the bot master. The collected information can be related to private users data, financial transactions, user passwords, etc. Botnets, that until recently were mostly related to personal computers, since 2010 also attack mobile devices. One example of mobile bots with a goal to propagate malware is Plankton, described in detail in Zhou and Jiang [2012], that appeared in 2011 and currently has more than 2000 different variants. In 2016 we have witnessed the biggest DDoS attack ever seen, as stated by *Internet Security Threat Report Volume 22* [2017], performed by Mirai botnet, that rendered unusable many leading websites, including Netflix, Twitter and PayPal. Mirai botnet was performed using IoT devices with poorly implemented security practices, that emphasized the need to increase security measures in IoT devices and protect them against similar threats in future.

- **Financial Malware** has a goal of collecting accounts credentials and sending them to the attackers. Current Android malware can intercept text messages with authentication codes from customers bank and forward them to attackers. Also, fake versions of legitimate banks mobile applications exist, hoping to trick users into giving them account details. Number of encountered attacks related to financial malware is increasing. This can be especially seen in the increase of banking malware, that attacks online

[1] `http://wapo.st/2pKyXum?tid=ss_tw&utm_term=.6887a06778fa`

banking customers. According to *IT Threat Evolution In Q2 2015* [2015], number of encountered banking attacks has increased from 71% to 83% from first to second quarter of 2015.

- **Logic Bombs** are pieces of code intentionally inserted into a software system that set off a malicious function only when specified conditions are met. When activated, a logic bomb can perform different actions: display spam messages, delete or corrupt data, execute pieces of malicious code or have other undesirable effects.

- **Viruses** are the type of malware that propagates by inserting themselves into another program and spreading together with it. The level of severity of viruses can vary from low, for example corrupting some files on the system, to high that can disable or even completely damage the operating system. Viruses are spreading together with the program they are attached to. It can happen by using Wi-Fi network, bluetooth, message or email attachments.

- **Worms**, as opposed to viruses that depend on a host program to be spread, operate more independently of other files. Still, same as viruses they are able to self-replicate and spread. In mobile devices, worms spread without users knowledge, by using existing communication channels: SMS, MMS, and bluetooth. First mobile malware, Cabir, that appeared in 2004 and was able to spread itself via bluetooth, was a worm developed for Symbian Operating System and ARM architecture. Since then different variants of worms exist in mobile devices, causing users information leakage, disruption of services or sending premium rate messages.

- **Trojans** are type of malware that appears as a legitimate software, but actually has malicious intents. Also, they are able to open a backdoor in a system, thus enabling further attacks. Due to their similarity with legitimate applications, detection of Trojans is a challenging task. At the same time, they are one of the most present malware type in mobile devices, especially devices running on Android Operating System. One of the most famous is Spitmo, a Trojan which steals information from the infected smartphone, monitors and intercepts SMS messages from banks and uploads them to a remote server, as described by Zhou and Jiang [2012].

Apart from malware, threats that can also appear in mobile devices are classified as grayware or madware. According to *Internet Security Threat Report Volume 20* [2015], out of the 6.3 million applications analyzed in 2014, one million were

classified as malware, while 2.3 million were classified as grayware. A further 1.3 million applications within the grayware category were classified as madware.

- **Grayware** are all the programs that do not contain viruses and are not obviously malicious but that can be annoying to the user, like for example adware (advertising-supported software), that automatically delivers advertisements.

- **Madware** consists of different aggressive techniques developed in order to place advertisement in mobile devices, for example photo albums and calendar entries and to push messages to notification bar.

Apart from the listed threats, there are various other forms of malware, grayware, and madware that have different names and different forms. Some examples are *phishing*, that is looking for someone to get "hooked" and load malware/grayware or madware and *spoofing*, that is pretending to be someone else (e.g., user's bank), win trust and exploit the credentials. Although the number of all possible threats that can happen in mobile devices is much higher, in this chapter we focus on and discuss in more detail the ones related to mobile malware, since it is currently the threat that can cause the most severe damage to devices.

When it comes to security of other IoT mobile devices, we have already seen attacks in ATMs, home routers, cars and medical equipment, but, according to *McAfee Labs Threats Report* [2015], these are just beginnings of attacks on IoT. Most of these devices connect via bluetooth that is known to suffer from many security flows, as stated in *2016 Threats Prediction* [2015]. Apart from being able to collect data stored on these devices, attackers can also abuse their connections to smartphones. Symantec in *Internet Security Threat Report Volume 20* [2015] discovered that 20% of applications related to health sent personal information, logins, and passwords over the wire in clear text.

In addition to the expansion of the existing attacks, also appearance of new ones is expected in the next years. According to threats prediction in *New Rules: The Evolving Threat Landscape in 2016* [2015], some of the threats that will become more aggressive and widespread in coming years are following: the rise of machine-to-machine attacks, propagation of worms in headless devices, and two-faced malware.

- **Machine-to-Machine Attacks** will take advantage of connected systems of mobile devices like connected medical devices and their host applications, connected home automation, smart TVs, and also connected home routers.

- **Worms in Headless Devices** refer to foreseen spread of worms within less complex devices, like smartwatches, by means of communication protocols.

- **Two-faced Malware** is type of malware designed to execute an innocent task to avoid detection system, and then, once it bypassed security checks, execute its malicious payload.

2.2 Existing Malware Detection Solutions

With increased number of mobile threats the need to protect from them is growing, resulting in higher demand for effective detection systems. CyberEdge Group [2015] indicates marked growth in the usage of anti-virus and anti-malware solutions for mobile platforms, which went from a 36% rate of use in 2014 to 45% in 2015. Although number of threats is observed in variety of mobile devices, most of existing malware is targeting smartphones. Due to this reason, most of current solutions, static and dynamic, are provided for them. In the rest of this section we discuss these solutions.

2.2.1 Static Detection

Static methods are focused on analysis of static features of applications (e.g., granted permissions, API calls, source code debugging) and discrimination between malware and benign applications based on this information.

One subgroup of static detection is signature-based, that is commonly used in current anti-virus and anti-malware solutions. It is based on the generation of representative signatures for existing malware samples and maintenance of a database consisting of them. Once the signature is recognized, malware is detected with high confidence. Although the number of false positives with such systems is low, they heavily rely on the maintenance of a database with signatures. Namely, it has to be frequently updated with new signatures that appear on the market. In mobile environment, this is difficult due to the fact that the device is not constantly connected to the Internet with high bandwidth, that sometimes is connected with mobile data that is charged, or that the device does not contain enough memory to store all available malware signatures.

The analysis of the effectiveness of the four representative mobile anti-virus software (AVG, Lookout, Norton, and Trend Micro) performed by Zhou and Jiang [2012], showed not so encouraging results of the detection performance of the

used software on the introduced Malware Genome dataset containing 1,260 malicious samples from 49 malware families; AVG detected 689 in 32 families obtaining detection accuracy of 54.7%; Lookout detected 1,003 malware samples in 39 families making it a detection accuracy of 79.6%; Norton was able to detect 254 samples in 36 families thus being only 20.2% accurate; and Trend Micro detected 966 samples in 42 families, obtaining thus accuracy of 76.7%. Based on this analysis, Zhou and Jiang [2012] stated that traditional content-signature-based approaches have been demonstrated not promising at all to cope with rapid growth and proliferation of malware.

Drebin, static approach to mobile malware detection, was proposed by Arp, Spreitzenbarth, Hubner, Gascon and Rieck [2014] where high detection accuracy is achieved by using features from the manifest file and feature sets from disassembled code. Using these features and Support Vector Machines (SVMs) classification algorithm, the authors perform the evaluation of the method on 123,453 applications, out of which 5.560 are malicious, and where the method was able to detect 94% of the malware with false positive rate of 1%. Detection is performed on the phone and the reported overhead is sub-linear as its overhead increases with $O(\sqrt{m})$, where m is the number of analyzed bytes.

DroidMat, the mechanism presented by Wu et al. [2012], also uses static features including permissions, Intent messages passing and API calls to detect malicious Android applications. The classification method that this system uses is kNN clustering and the detection is performed on the server, reporting linear overhead in the size of the problem. For the evaluation of the approach 1,500 benign and 238 malicious applications were used with the reported precision of 0.9674, recall of 0.8739, and F-measure of 0.9183.

DroidAPIMiner is another approach to static malware detection, presented by Aafer et al. [2013]. It is a machine-learning based approach to Android malware detection that is taking into account only features from API calls. The detection performance of this method was evaluated on a dataset consisting of 20,000 applications, out of which 3,987 malicious, and the highest achieved detection accuracy is of 99% and a false positive rate is of 2.2%, that was achieved using kNN classifier. In addition to kNN, the authors also experimented with two decision trees based algorithms, and SVMs, but kNN outperformed them in terms of accuracy and true positives rate.

FlowDroid, presented by Arzt et al. [2014], is a static taint analysis based tool, that models applications life-cycle and callback methods and using these models detects sensitive privacy leaks in Android applications. The method is evaluated on Stanford SecuriBench Micro Livshits [2013], a set of micro benchmarks originally intended for web-based applications, and on DroidBench, proposed by the

authors. The achieved detection performance on DroidBench is precision of 86% and recall of 93%, on SecuriBench Micro reported recall is 96% and reported false positives are nine.

In the AppContext, introduced by Yang, Xiao, Andow, Li, Xie and Enck [2015], the authors propose to discriminate between the malicious and benign behaviors looking at the contexts that trigger security-sensitive behaviors. They start from a set of actions previously defined as suspicious and then add a context by considering which category of input controls the execution of such suspicious actions. The AppContext is evaluated on a dataset consisting of 202 malicious applications and 633 benign applications from the Google Play Store, and it achieves detection accuracy of 95% and recall of 87.7%.

Truong et al. [2013], as a feature for detecting susceptibility of a device to malware infection, use a set of identifiers representing the applications on a device is used. The assumption is that the set of applications used on a device may predict the likelihood of the device being classified as infected in the future. The authors, that used Multinomial Naive Bayes classifier on 103,695 applications, concluded that although the set of other applications installed on the phone might be an indicator for the presence of malware, observing just this feature is not enough to give precise answer about device being attacked.

Kirin, presented by Enck et al. [2009], is an analysis tool that performs Android malware detection by flagging an application as suspicious according to a set of predefined rules based on the combinations of requested permissions. Also Apple, Google, and Nokia use application permissions and review to protect users from malware. The effectiveness of these mechanisms against malware is evaluated by Felt, Finifter, Chin, Hanna and Wagner [2011], where sending SMS messages without confirmation or accessing unique phone identifiers like the IMEI are determined as promising features for malware detection as legitimate applications ask for these permissions less often, as stated by Felt, Greenwood and Wagner [2011]. Also checking if groups of certain permissions are present is shown to be useful in the detection of malware. For example, nearly one third of applications request access to user location but far fewer request access to user location and to launch at boot time. Features derived from permissions usually do not provide good detection performance of malware, also because malware writers often use a technique called *overpermission*; by using this technique, applications ask for a broad range of permissions, even though they would not be required by the tasks that they perform; additional permissions are exploited by dynamically loaded malicious payloads.

The analysis of disassembled code, more precisely of opcodes, is proposed by Canfora, Mercaldo and Visaggio [2015], where features related to the op-

code occurrences are considered. More precisely, the authors classify malware using a set of features which count the occurrences of a specific group of opcodes extracted from the smali Dalvik code of the application under analysis. Smali is the language that represents the disassembled code for Dalvik Virtual Machine. The proposed approach is evaluated on a dataset containing 11200 applications on which it showed detection precision of 93%. The considered opcodes, that have been chosen as representative of the alteration of the control flow, are following six: *move*, *jump*, *packed-switch*, *sparce-switch*, *invoke*, and *if*. Additionally, the method proposed by Canfora, De Lorenzo, Medvet, Mercaldo and Visaggio [2015] demonstrates that the sequences of opcodes are effective in detecting Android malware. In this approach, the authors consider a binary classification problem in which an input application a has to be classified as malware or benign using the occurrences of two opcodes as features (i.e., 2-grams). The authors evaluated the approach on a dataset composed of 11,120 applications, out of which 5,560 were malware. The used classification algorithms were SVMs and Random Forest (RF), out of which RF performed better on the observed task and obtained detection accuracy of 96.88%. RF is a combination of different tree classifiers as discussed by Breiman [2001].

In the nutshell, static detection methods are effective in terms of resource consumption. Usually they are less computationally intense than dynamic methods and that they do not need applications to be run for identifying the infection according to Martinelli et al. [2016]. They are also fast when automated tools are used. Additionally, since the analysis can be performed offline and without need to execute programs and collect the traces of behavior, it is more convenient for large scale experiments with high number of applications, both benign and malicious, that gives more confident results about the detection performance of the proposed systems. Such large scale experiments, in dynamic scenario where applications have to be executed and run for some time, are not realistic, and thus usually smaller datasets than in static scenarios are used. The main problem of static analysis is that it is relatively easy to create a new malicious sample that would be able to evade detection system based only on it. It is shown that it is prone to obfuscation and cannot detect variations of existing malware samples that are easy to create and distribute according to Moser et al. [2007]. Additionally, due to the nature of this approach that analyses the applications only based on their static features, it is not able to detect malware that appears at runtime. In this way users stay unprotected at runtime, in scenarios when they visit a website that might contain malicious code, open malicious attachments, when they click on a pop-up window, or when already installed applications perform malicious update.

2.2.2 Dynamic Detection

Dynamic detection is a promising candidate for mobile malware detection systems since it is able to detect variety of malicious samples that currently exist on the market and is more resistant against previously unseen malicious samples coming from existing malware families. The main advantage of this approach is that dynamic system features are observed at runtime, such as for example system calls and network behavior, and based on them and previously trained models, detection is performed. In this way, by observing the behavior of the system at runtime, systems are more resistant to variety of existing malware samples and more difficult to bypass. The reasoning behind is that while attackers can obfuscate the code itself it is difficult to obfuscate its behavior. Additionally, if attackers tried to evade detection by encrypting some parts of the application source code, such trial will not be of use in case of dynamic detection in contrary from static detection.

Dynamic detection mechanisms are used by Bose et al. [2008] to detect mobile worms, viruses and Trojans. The authors start with the extraction of representative signatures. Later on, a database with malicious patterns is created and SVMs are used in order to train a classifier with both benign and malicious data. The evaluation of both emulated and real-world malware shows that dynamic detection not only results in high detection rates but also detects new malware which shares certain similarity with existing patterns in the database. The evaluation of the real-world performance of the method is performed on two, at that time, available Symbian worms: Cabir and Lasco.

Power consumption, monitored through battery usage, is also used as a dynamic feature to indicate the presence of mobile malware, as introduced by Becher et al. [2011]. One of the proposed solutions, VirusMeter proposed by Liu et al. [2009], monitors and audits power consumption on mobile devices with a power model that characterizes power consumption of normal user behaviours. In order to evaluate the effectiveness of the approach, the prototype is developed on Nokia 5500 Sport and evaluated against malware samples FlexiSPY and Cabir, that were detected by the proposed approach with less than 1.5% additional power consumption in real time. Kim et al. [2008] propose creation of a database with power signatures, where a new power signature collected while the system is used is compared with the ones already existing in the database. This evaluation was performed using anomaly detection methods on Windows mobile and for it four malware worms and one battery-depletion attack were used. Although in the observed scenarios, where small number of malicious samples is evaluated, battery usage related detection showed promising results,

to what extent modern malware can be detected on phones by monitoring just the battery power, as stated by Becher et al. [2011], remains an open research question.

SmartSiren, presented by Cheng et al. [2007], is a collaborative virus detection and alert system for smartphones. It performs statistical and abnormality monitoring, detects abnormalities at both device and network level, and in case alerts being detected issues alarm to the targeted population. Rather than focusing on malware, the approach is tested and validated, via simulation, on viruses spreading via Bluetooth and SMS and Windows Mobile 5.0 Smartphone Edition. The dataset used consists of three weeks of SMS traces collected from Indian national cellular service provider and the implementation of two viruses Cabir and Flexispy. The reported overhead is 33.6% of the total messages.

Schmidt et al. [2007] present the approach to identify the most representative features to be observed on a phone running on Symbian Operating System and then sent to the network so that anomaly detection is performed there. Following five features are identified as informative: RAM Free, User Inactivity, Process Count, CPU Usage, SMS Sent Count. The authors validated the importance of the features by using as a dataset simulation of normal behaviour of 10 frequently used applications at that time, and one malware sample.

Xie et al. [2010] propose a probabilistic approach to detection of malware propagating through Bluetooth and messaging services. It observes unique behaviors of the mobile applications and the operating users on input and output constrained devices, and builds a Hidden Markov Model to learn application and user behaviors. Later, based on this knowledge, it identifies behavioral differences between malware and human users. The analysis, for which 346 SMS sequences of benign users and 27 abnormal sequences, is performed on Linux-based smartphone, using 16 features.

Shabtai et al. [2012] propose Andromaly, a framework for detecting malware that uses variety of features related to: touch screen, keyboard, scheduler, CPU load, messaging, power, memory, calls, operating system, network, hardware, binder, and LEDs, and compares False Positive Rate, True Positive Rate, and accuracy of the following detection algorithms: Bayes Net, Decision Tree J48, Histogram, K-means, Logistic Regression (LR), and Naive Bayes (NB). The algorithms that outperformed the others in detection of Android malware were LR and NB. The results were obtained using 40 benign applications and four developed malicious samples, since no real malicious applications were available at that moment.

Ham and Choi [2013] perform feature selection on a set of run-time features related to network, SMS, CPU, power, process information, memory and Virtual

memory. As a measure of features importance, Information Gain was used along with four classification algorithms: NB, RF, SVMs, and LR. Results show that, in this scenario, RF gives the best performance. Results have been obtained by considering 30 benign and five malicious applications.

Spreitzenbarth, Freiling, Echtler, Schreck and Hoffmann [2013] present an automatic way to detect malware by using combination of static and dynamic approach towards malware detection. In order to extend coverage of dynamic detection, static detection is used as a first step, where the authors take into account applications Manifest file, decompiled code and requested permissions. Further, they analyze the application in sandbox tracking native API calls of the application taken into account. Malware samples taken into account are 136,000 applications from Asian and Google Play market and 7,500 malicious samples. The system is not envisioned to be run on a mobile device, but instead is accessible via web interface for all the users that would like to test the suspicious applications.

Work proposed by Burguera et al. [2011], is a crowdsourcing system that uses real traces of application behaviour collected from users. The traces are analysed in the network by usage of k-means clustering. Malware is detected by investigation of system calls, and the authors argue that the monitoring system calls are the most accurate way to detect malicious Android applications, since they provide detailed overview on the events. Dataset used is consisting of Trojan samples, more precisely, three samples of self-written malware and two real malware application samples.

Dini et al. [2012] propose Madam, a Multi-Level Anomaly Detector for Android Malware. It is a framework that detects intrusions and malicious actions on Android devices. It does the detection by monitoring system OS events (system calls) and the user activity/idleness. The evaluation of the system is performed by using 10 real malware samples on Android Ice Cream Sandwich Version 4.1 Samsung Galaxy Nexus phone. Using thirteen features for the detection the accuracy of 93% was obtained. The reported overhead of the approach is 3% of memory consumption, 7% of CPU overhead, and 5% of battery. In order to use Madam, rooting of a phone is required.

A work proposed by Oberheide et al. [2008] consists of two components: a host agent and a network service. The host agent is acquiring files and sending them to the network service, whereas the network service performs analyses using multiple detection engines in parallel to determine whether a file is malicious or not. The evaluation of the approach is performed on two Nokia phones N800 and N95, and it showed that the proposed approach consumes less memory and CPU due to the offloaded detection. However, as also stated by the authors, since

the detection is completely offloaded from the phone and done on the server, it rises privacy issues and also leaves users unprotected in scenarios where they are disconnected from the network.

Another solution in which the detection is completely offloaded in the network is ParanoidAndroid, proposed by Portokalidis et al. [2010], that uses the anomaly detection principle. Based on phone execution traces, security checks are performed on the synchronized copy of the phone that runs on a server. The phone used in the evaluation of the method is HTC G1 phone, and on the server side QEMU was used. The results show that using such approach battery life is reduced by about 30% and CPU load by about 15%.

Many detection approaches have been developed which work on features derived from system calls, or sequences of system calls, occurrences or frequencies. CopperDroid, proposed by Reina et al. [2013], recognizes malware samples through a system calls analysis with a customized version of the Android emulator in order to track system calls. In Wang et al. [2009], an emulator is used to perform a similar task: the method is assessed on a dataset composed of 1,600 malicious applications; 60% of the malicious applications belonging to the Genome Project as well as the 73% of the malicious applications included in the *Contagio Dataset* [2016] are identified correctly.

Canfora et al. [2013] propose a method to detect Android malware based on three metrics, which evaluate the occurrences of a reduced subset of system calls, a weighted sum of a subset of permissions required by applications, and a set of combinations of permissions. In their experiment a sample of 200 malicious applications and 200 benign applications are considered, and a 74% precision in the identification of malware is obtained. Canfora, Medvet, Mercaldo and Visaggio [2015] propose also another approach to malware detection based on system calls. The approach uses machine learning to discover connections between malicious behavior (e.g., sending high premium rate SMS or ciphering data for ransom) and their execution traces and then exploit obtained knowledge to detect malware. As opposed to other systems, where a limited set of system calls is taken into account, in this work, all system calls are considered so as their sequences. The approach is tested with data coming from a real device, with a dataset consisting of 20,000 execution traces and 2,000 applications and obtains a detection accuracy of 97% (on previously unseen applications 94.9%). In order to detect malware, 750 features related to system calls are taken into account. Although the achieved detection accuracy is high, the detection time of this approach is not reported, neither the resource consumption on a device.

The detection method presented by Isohara et al. [2011] uses data gathered by an application log and a recorder of a set of system calls related to manage-

ment of file, I/O and processes. A physical device with a modified Android 2.1 was used for the experiments and 230 applications, in greater part downloaded from Google Play, were considered; among them, the method is able to detect 37 applications which steal personal data, 14 applications which execute exploit code, and 13 destructive applications. For this method the modification of the kernel is needed.

Monitoring of system calls in Android is also discussed by Bläsing et al. [2010]. The authors perform both static and dynamic analysis, disposing a module which monitors system calls and logs the return value of each system call. The system is not evaluated on devices, since it is created for analyzing Android applications via cloud service.

Dimjaševic et al. [2016] propose a method to perform automatic classification based on tracking system calls while applications are executed in a sandbox environment. The analysis is performed on 4, 289 malicious applications from Drebin dataset proposed by Arp, Spreitzenbarth, Hubner, Gascon and Rieck [2014] and on 8.371 benign applications from Google Play Store. The approach relies on 15.000 features, and using them trains RF, SVMs, LASSO regression analysis, and ridge regression. The best results are obtained using RFs, and the accuracy of 93% with a 5% benign applications classification error is achieved. The system is developed relying on the Android Emulator, and its detection time and resource consumption is not reported.

The increasing number of malicious infections by ransomware is rising concerns about the effectiveness of current malware detection methods in their detection. Due to this, some of the recently proposed mobile malware detection methods, that we discuss in the remainder of this section, are proposed specifically for the ransomware detection scenarios.

The first method that introduced ransomware detection for Android is HelDroid proposed by Andronio et al. [2015]. This tool includes a text classifier based on NLP features, a lightweight smali emulation technique to detect locking strategies, and the application for detecting file-encrypting flows. The main weakness of HelDroid is represented by the fact that it strongly depends on a text classifier: as a matter of fact, the authors trained it on generic threatening phrases, similar to those that typically appear in ransomware or scareware. This strategy can be easily thwarted by means of techniques such as string encryption and data ciphering introduced by Rastogi et al. [2013]. Furthermore, the proposed method strongly depends on language dictionaries; this is the reason why, as stated by the authors, when the analyzed ransomware is targeting non-English speakers, the dictionary must be switched to a different language. In addition, this method can be evaded by altering the occurrences of such words. For the

performance point of view, they identify rightly 375 Android ransomware on a dataset composed of 443 samples: 11 ransomware were not detected due to unsupported language (e.g., Spanish, Russian) with 9 out of 12,842 false positives.

Yang, Yang, Qian, Lo, Qian and Tao [2015] propose a performance tool in order to help to understand what can be done to cope with Android ransomware detection. This tool provides the ability to dump the log of system messages, including stack traces. However, this method remained at the level of a proposal, with no implementation. Therefore, there are no results that can prove its effectiveness.

Song et al. [2016] designed an approach with the aim of identifying mobile ransomware by using process monitoring. They consider features representing the I/O rate as well as the CPU and memory usage. They evaluate the proposed method with only one ransomware sample developed by the authors, this sample has the ability to encrypt the file by using AES.

An approach based on formal methods that is able to detect Android ransomware and to identify the malicious sections in the application code is described by Mercaldo et al. [2016]. The authors evaluate a dataset composed of 2,477 samples with real-world ransomware and benign applications. Starting from the payload behavior definition the authors formulate logic rules that are later applied to detect ransomware. The main weakness of the proposed method is represented by the human analyst effort required to build the logic rules: as a matter of fact the proposed method foresees the payload identification but the process rule building has to be done by hand, and as such, is time consuming task.

As it can be seen from the previous description, several detection approaches based on dynamic analysis of resources have been proposed for mobile devices. They can be divided with respect to the types of detection techniques used, detection side (mobile device or cloud), operating systems, used datasets, so as computational overhead of the used methods. In Milosevic, Regazzoni and Malek [2017] we divide the existing approaches with respect to these characteristics. Additionally, trustworthiness of mobile devices involves trustworthiness of both hardware and software components. Trustworthiness of hardware is achieved by addressing threats such as hardware Trojans or physical attacks, while software security is generally affected by malware. While in this chapter we summarize current security threats and existing solutions for mobile devices from the software perspective, in Milosevic, Regazzoni and Malek [2017] we address security problems also at hardware level.

According to Rastogi et al. [2013], dynamic methods are able to discriminate malware even when its code is obfuscated. However, they need applications to

be run to identify malicious behaviour, potentially infecting the device, as stated by Martinelli et al. [2017]. The other drawback of dynamic detection methods is that such systems might be too complex for limited resources of mobile systems. In some cases, as previously mentioned, detection engines are offloaded to a cloud or a server, thus imposing new challenges to the system related to data transmission, communication overhead, data privacy, and lack of protection. Additionally, most of the proposed work, apart from detecting the presence of malware, is not able to identify what type of malware is being executed nor which parts of the malicious programs are actually malicious. Finally, detection time and consumed power, that are one of the main challenges for the successful deployment of dynamic detection methods at runtime, are in most cases not reported along the obtained detection accuracy.

In order to extend the state of the art, we propose a detection approach that takes into account constrained environment of mobile devices from its early design and is suitable for on-device runtime usage and monitoring against malicious activity. In addition, our detection method investigates what parts of applications are actually malicious and provides results on malware families to which the running application potentially belongs. Finally, it takes into account the detection time as one of the main requirements and focuses on providing an early decision about the possible infection by malware. More detailed comparison of our approach with state-of-the-art methods is given in Section 5.6.

Chapter 3

Proposed Methodology for Malware Detection at Runtime

In this chapter, we describe the methodology that we propose in order to perform both effective and efficient on device, at runtime, defense from mobile malware. Our main idea is to design a malware detection and malware classification systems that are optimized for detection accuracy, detection time, and battery power consumption. Each of these requirements is important for providing effective and efficient defence against mobile malware, but up to date, to the best of our knowledge, they have never been considered in such a comprehensive way in the existing literature. In order to achieve good detection accuracy, we look at the behavior of both malicious and benign applications as reflected by operating system parameters related to its dynamic behavior (memory, CPU, system calls and network behavior) and we identify the parameters that are the most indicative for the presence of malware. Detection of malware is performed in two steps, at execution records level first, and then at the application level. In this way, we can understand what parts of executions are malicious in addition to which applications are malicious. In order to minimize battery power consumption, we decrease the number of observed system features to only the most indicative ones and we use detection algorithms of low complexity suitable for mobile environment. Additionally, we investigate how important it is to select the most suitable sampling period of monitoring infrastructure for the detection time, detection accuracy and consumed power. Together with the most indicative features for the detection of malware, during programs execution we also record the behavior indicative for presence of different malware families, needed for malware classification task, and in case an alarm is triggered by detection module, we send this information to the network for further analysis. In the remaining part of this

chapter, we discuss the main components of the proposed detection system and their characteristics.

3.1 Overview of Proposed Methodology

The proposed approach is analogous to general practitioner versus specialist approach to dealing with a medical problem, as we also discuss in Milosevic, Ferrante and Malek [2015]. Similarly to a general practitioner who, based on indicative symptoms identifies potential illnesses and sends the patient to an appropriate specialist, our detection system detects potential presence of infection with malware and, once the symptoms are detected, it sends information needed for specific analyses about which infection it is. The approach we propose in order to perform this task consists of two parts: runtime detection, that is aimed to be used on mobile devices, and offline development, that is aimed to be performed offline on devices with more computational capabilities than mobile. In Figure 3.1 we depict the outline of modus operandi of the proposed approach, and in Figures 3.2 and 3.3 the offline development of malware detection and malware classification components, respectively.

As it can be seen in Figure 3.1, in our approach to malware detection, in order to detect malicious records, a set of the most indicative features are monitored and analyzed by using detection algorithms suitable for resource-constrained devices. In order to detect malicious applications, we consider the history of execution records (marked as malicious or not). The features that are observed in this module are related to dynamic aspects of applications behavior at runtime. These features are easy to collect in terms of computational resources and at the same time they are informative about the device behavior. In the offline phase, depicted in Figure 3.2, it can be seen that after executions of the considered malicious and benign application, we extract all the features of interest, perform the feature selection methods on them to detect only the most indicative ones, and then, using only the selected features we perform the training of classifiers.

As shown in Fig. 3.1, at runtime we store a set of features related to malware classification and when an alarm is raised we send it to a remote server for further analysis. The offline development of the classification module is depicted in Fig. 3.3, where it can be seen that, after the execution of applications and collection of their traces, we derive the set of the most indicative features for each observed family and validate their importance using a suitable classifier. Then, at runtime on the mobile device, we store a set of all the features identified as the most indicative for observed families and, when an alarm is raised, we send

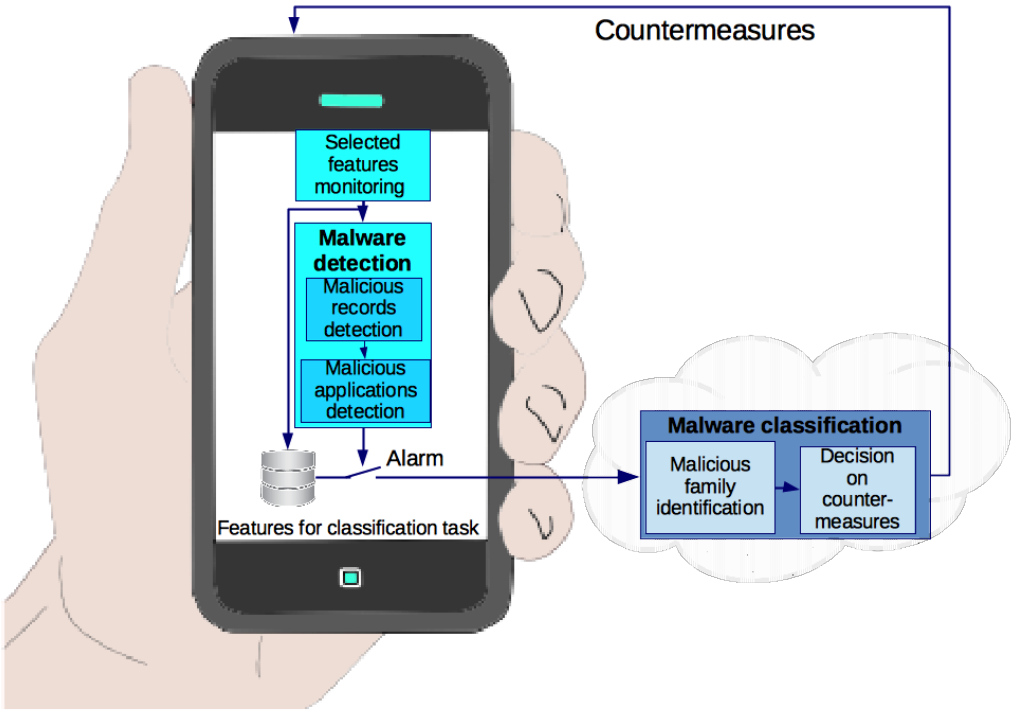

Figure 3.1. Malware detection and malware classification system.

it into the network. In the offline phase different classifiers are tested and the most suitable ones for observed families can be identified to be used for their later detection. As a quantitative description of the quality of detection methods, we use F-measure (sometimes also called F-score), that is a harmonic mean of precision (the fraction of detected instances that are relevant) and recall (the fraction of relevant instances that are detected) and as such conveys the balance between these two important aspects of the detection system. In addition, most of the state-of-the-art methods also take into account F-measure and report the obtained detection results with respect to it, thus the use of F-measure in our scenario facilitates comparison to the related work.

The approach we propose in this thesis deals with detection of malicious software and as such is designed, tested and validated with respect to its detection performance and classification of malicious applications. Due to a potential similarity of maliciously introduced errors and the errors that could happen by accident, it might happen that our detection approach can identify up to a certain point also accidental errors. However, due to the fact that the accidental errors are out of the scope of this thesis, we did not perform any evaluation of the ef-

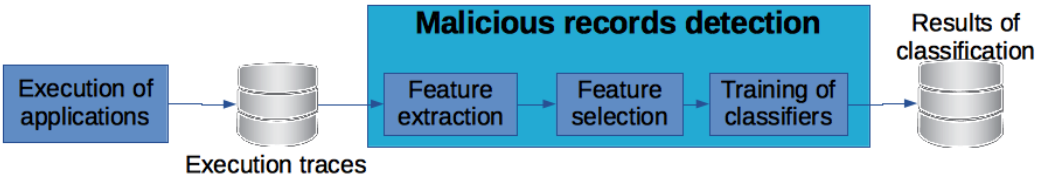

Figure 3.2. Offline development of the malware detection mechanism.

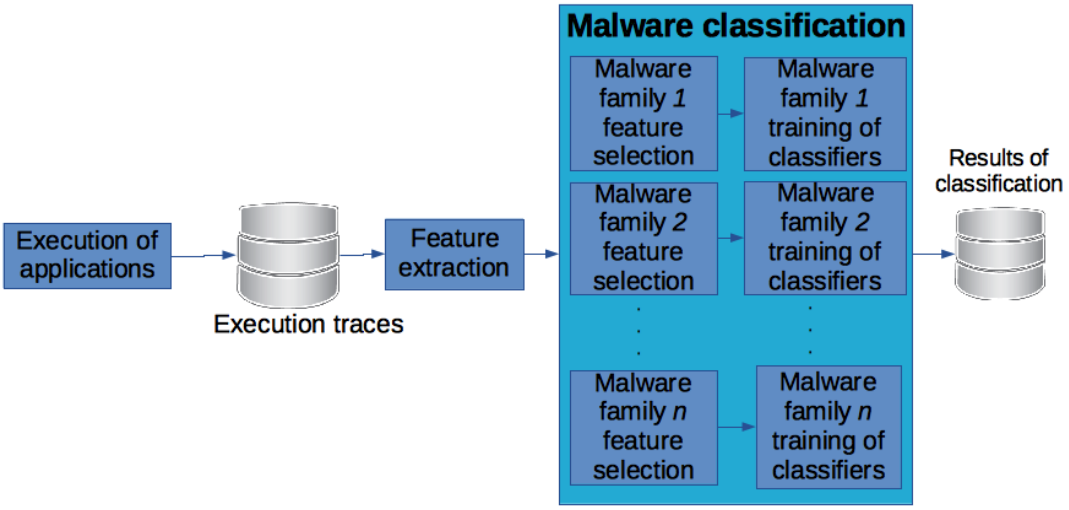

Figure 3.3. Offline development of the malware classification mechanism.

fectiveness of our approach with respect to their detection. Similar situation is with comparison between failure prediction methods and our approach. Namely, having in mind similar symptoms of failures and malicious attacks, it can happen that the approach we propose is able to identify also other types of failures not caused by malware, but due to the fact that a quantitative analysis of the detection performance of our method is performed only in the case of malware, we cannot provide a quantitative measure of this effect.

3.2 Selected Features Monitoring

Systems based on dynamic features are more resistant to the obfuscation than the ones based on static ones and are able to detect wider range of malware, as discussed in Chapter 2. In order to achieve good quality of malware detection represented in high F-measure, and thus high effectiveness of the approach, we

use dynamic features related to the resource usage (memory, CPU, and network) and system calls that reflect well the behavior of the program Ferrante et al. [2016]. Simultaneously, the key enabler for the efficient detection on the phone is monitoring of only a limited set of features at runtime. Decreasing number of features lowers both the complexity of monitoring infrastructure and complexity of detection algorithms influencing thus overall consumption of resources on mobile devices. Due to these reasons, selection of a set of indicative features that contain crucial information to discriminate between malicious and benign applications on one hand, and are limited in number and can be used in resource-constrained environment on the other, is a challenging task and the successful identification of such a set is one of the most significant parts of the proposed approach.

We perform the indicative features selection for both the malware detection and malware classification tasks. The initial obtained results on the influence of the observed features for both tasks is discussed in Section 5.1. These features and their verification in the malware detection domain is explained in Section 5.2.1 and in the malware classification domain in Section 5.3.

3.3 Malware Detection

Our goal is to perform effective and efficient detection of both malicious parts of the applications and the complete malicious applications, while taking into consideration detection time. The outline of the envisioned malware detection system, and the main components of the proposed detection systems, is given in Figure 3.4.

Runtime detection consists of the four components: *Selected Features Monitoring*, *Malicious Records Detection*, *Malicious Application Detection*, and *Alarm*. *Selected Features Monitoring* monitors suitable system parameters with a predefined sample rate and extracts features from them. *Malicious Records Detection* is used to categorize every execution record as malicious or benign. *Malicious Applications Detection* classifies a complete application as malware or not. The *Alarm* is used to raise an alarm in case a malicious application is found. This module can be used to alert the user as well as to activate suitable countermeasures in the device. However, the main purpose of this module in our scenario, is to inform an external cloud infrastructure about an infection potentially taking place, and send the stored indicative features for malware classification task, in order to be used to identify which malware family is attacking the device.

The training and the development of the components to be used at runtime is

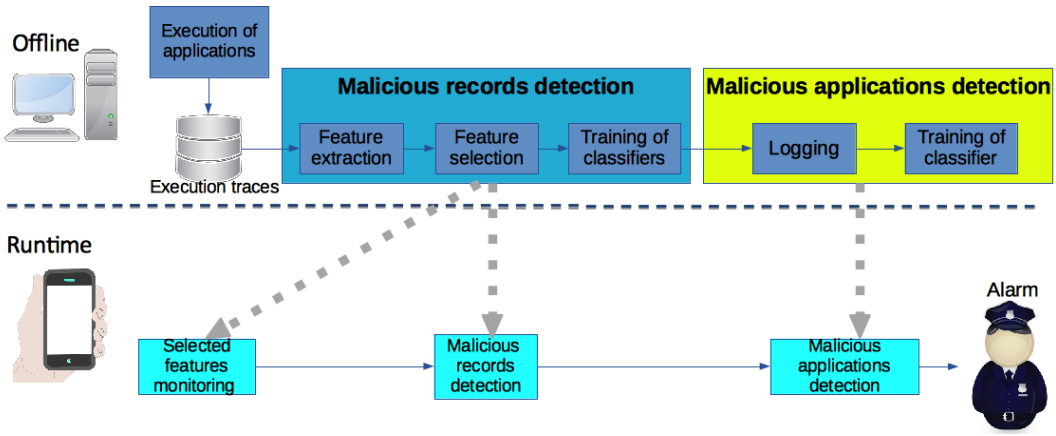

Figure 3.4. The proposed detection system's offline blocks and its runtime components.

performed offline to guarantee the possibility of repeating experiments with different algorithms and parameters: the features to be monitored by the *Selected Features Monitoring* block are identified by using a set of feature selection methods on previously recorded execution traces; the algorithm and the parameters to be used in the *Malicious Records Detection* and in the *Malicious Applications Detection*, are trained by using the same recorded execution traces as input. Therefore, it is necessary to execute applications, collect execution traces, and to extract features: these actions are performed by the *Execution of Applications* and *Feature Extraction* components, respectively. These components, together with the main parts of the developed methodology, are described in the following part of this chapter.

3.3.1 Malicious Records Detection

The main goal of the malicious records detection is to distinguish between execution records belonging to malicious and benign applications, using at the same time the detection algorithm compatible with the limited resources of mobile devices.

While most of the steps of the proposed approach are executed offline, the detection algorithm is executed on the mobile devices; thus, it needs to be compatible with their limited resources. This is the reason why, for this task, we take into account the complexity of algorithms, and only consider the ones with low complexity. A brief description of classification algorithms that are potentially suitable for the envisioned scenario follows:

- *Naive Bayes (NB)* is a probabilistic classifier. It applies Bayes theorem making an assumption that the features are independent. Under this *naive* assumption it calculates probability of an unknown instance belonging to each class and selects the one with the highest probability as an output, as stated by John and Langley [1995].

- *Logistic Regression (LR)* is a linear classifier that calculates the conditional probabilities of possible outcomes and chooses the one with the maximum likelihood. Le Cessie and Van Houwelingen [1992] motivate its usage with ridge estimator in order to further improve its prediction capability.

- *A decision tree-based J48 classifier (J48)*, as stated by Hall et al. [2009], is an open source implementation of C4.5 algorithm, a statistical classifier that for each node of the tree, chooses the attribute of the data that the most effectively splits its samples into subsets; the splitting criterion being maximum information gain as introduced in Quinlan [1993].

The obtained results and performance of used malicious records detection are presented in Section 5.2.2.

3.3.2 Malicious Applications Detection

In order to detect malicious applications running on a phone, it is needed to observe behavior of execution records on the phone for certain amount of time and then identify them as such. To perform this task, we propose to train a detection algorithm named *Malicious Applications Detection*. We have designed a method where, by keeping only a limited history of past execution record classifications, we are able to identify malicious applications effectively. The method is based on a sliding window mechanism, observation period that changes ("slides") over time. Namely, considering a sliding window of length n, the percentage of records classified as malware in the last n instants of time is used to determine whether an application is malware or not. To make the mechanism more robust, multiple results, obtained in disjoint sliding windows, are considered and only when w windows are marked as malware, the application is classified as malware. Figure 3.5 shows an example of this algorithm where a malicious application is identified at sample 12, after the second (as $w=2$) window of length $n=4$ is marked as malware. In this example, the threshold on the number of malware records contained in each window is set to 70%.

The system proposed in Figure 3.4, is modular and each element can be changed independently from the others, provided that the interfaces are pre-

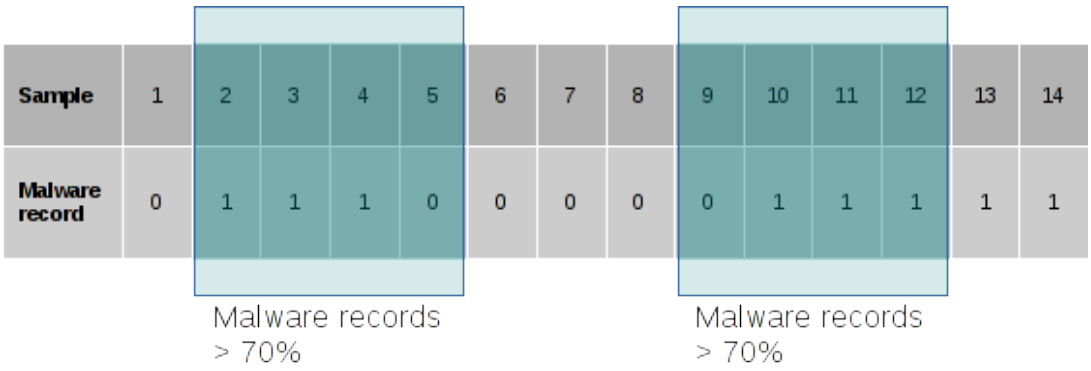

Figure 3.5. An example of the sliding window algorithm applied to a sequence of classified execution records; window length, threshold, and number of checks are set to 4, 70%, and 2, respectively.

served. In particular, algorithms for malicious records and malicious applications detection can be changed without affecting the general functionality of the detection mechanism; when the malicious records detection mechanisms is changed, a tuning of malicious applications detection parameters (window length, threshold and number of checks) may be necessary to obtain the best possible results. The monitoring mechanism can be changed without affecting other parts of the system, provided that the features monitored remain unchanged.

The results we obtained using the proposed method and a malicious applications classifier as a standalone method are given in Section 5.2.4.

3.3.3 Detection of Malicious Sub-traces

While the previously discussed malicious records detection is used to classify execution records, detection of *sub-traces* is related to the detection of groups of execution records. The approach we propose jointly uses several dynamic features related to system calls, memory usage, and CPU usage, to identify those groups of records in which an application behaves maliciously. We use system timestamps to connect our dynamic features, thus having knowledge about the behavior of different parts of the system at any given time. While in the scenario of malicious record detection, discussed in Section 5.2.2, we train classifiers using single execution traces, in this scenario we use groups of execution traces originating from both malicious and benign applications.

In the state of the art it is shown that high detection accuracy can be achieved using system calls. However, using system calls for identification of malicious sub-traces is a challenging task due to complexity of their analysis and to the dif-

ficulty of identifying anomalous parts in sequences of system calls consisting of system call names and their attributes. In order to overcome this problem, and to identify which parts of system calls traces are actually anomalous, we connect them with memory and CPU features, that are real-valued numbers and can be easily grouped into clusters with similar behavior. In this way, by having clusters of similar memory and CPU behavior, we have added also system calls of similar behavior, being able to further investigate them. Based on the information of collected system calls, we train classifiers in the way it was proposed in Canfora, Medvet, Mercaldo and Visaggio [2015] and then, during the application executions, by observing their memory and CPU behavior, we classify the sub-traces with learned classifiers. The main steps of the proposed methodology, consist of the application execution, the sub-traces collection, the sub-traces clustering based on similarity between memory and CPU information, and the learning of one classifier for each cluster.

In our method, given an execution of an application, a *trace* $t = (o_1, o_2, \dots)$ of observations is available. Each *observation* o is a set composed of:

- A timestamp,
- Values for CPU usage (total, user, kernel)
- Values for total memory usage (PSS, shared, private, heap allocation, free heap)
- Values for Dalvik memory usage (PSS, shared, private, heap allocation, free heap)
- Values for Native memory usage (PSS, shared, private, heap allocation, free heap)
- A sequence s of system calls generated in the time frame delimited by the current observation timestamp and the previous observation timestamp

The set of the 63 system calls has been used, that is chosen statically by selecting the most occurring system calls performed by Android applications (as proposed by Canfora, Medvet, Mercaldo and Visaggio [2015], where they provide high detection accuracy). Names of considered system calls are listed in Section 5.2.3. The learning set is composed by pairs (t, l) where T is a trace and $l \in \{\text{benign}, \text{malicious}\}$ is a label which is benign if the application for which t has been obtained is benign, and malicious otherwise.

The main idea is to divide a trace in sub-traces and to classify each of them by considering system calls. For this purpose, different classifiers are trained for sub-traces that exhibit specific CPU and memory behaviors.

The method itself consist of a *learning phase*, in which the learning set $\mathscr{L}$ is used to tune several inner parameters, and a *classification phase*, in which a single trace t is analyzed.

The aim of the learning phase is twofold. First, to cluster sub-traces of the labeled applications according to the corresponding behavior of the application in terms of CPU and memory usage. Second, for each cluster, to train a classifier based on system calls, so that it should be able to discriminate benign and malicious sub-traces.

The goal of the classification phase is to give an indication on sub-traces t' of traces t lasting time T about whether each t' is related to a benign or malicious behavior during the corresponding interval. To achieve this goal we first obtain the list of sub-traces. Then, for each sub-trace, we obtain the classification results from three nearest neighbours. The outcome of the classification is cast as a value in $[0,1]$ which corresponds to the probability, that the sub-trace has a label $l = \text{benign}$. After having the three classification outcomes, we combine them by means of a weighted average in which the weights are determined by the corresponding cluster accuracy. The underling motivation is to give more importance to an outcome generated by a classifier which has been trained on a cluster for which the difference between malicious and benign behaviors is sharper.

The results we obtained using the proposed approach to malicious sub-traces detection are listed in Section 5.2.3.

3.3.4 Optimization of the detection solution by changing sampling period of the monitoring infrastructure

The power consumption of developed methods is strongly related to the complexity of the monitoring infrastructure in addition to the complexity of used detection algorithms. In order to decrease this complexity, we focus on reducing the number of monitored features and usage of only the most indicative ones. However, an additional way to limit power consumption is through changing sampling period. In order to achieve this goal, we start by evaluating system performance when the sampling period is 2s, that is the minimum sampling period that we can achieve with our current experimental setup, and then we also investigated other meaningful sampling periods: 4s, 6s, 8s, 12s and 16s. In our current setup the maximum sampling period that is compatible with the duration

of the applications executions in our dataset and with the parameters considered in our sliding window based detection algorithms is of 16s. The results we obtained performing this evaluation are discussed in Section 5.2.5.

3.3.5 Application-specific usage of the proposed methodology

In this thesis, we focus on design, development and validation of a malware detection system that takes into account detection precision and recall reflected in F-measure, detection time and battery power consumption. In order to facilitate comparing different solutions with respect to the multiple considered parameters, we use a metric that summarizes these parameters. In our case, we opted for a simple metric, shown in Equation 3.1, that includes F-measure, power, and detection time with equal importance.

$$M = \frac{F - measure}{Normalized\ power \times Normalized\ detection\ time} \tag{3.1}$$

The results we obtained with respect to this metric and the analysis of the optimized configurations are described in Section 5.2.5. Although in our scenario we consider the three mentioned requirements as if they are of the same importance, we are aware that in different application-specific scenarios this can change, and that in some domains the high detection accuracy is of utmost importance, in some early detection has absolute priority, and in some the focus is on the low power consumption. Even though we do not consider these three extreme scenarios in detail, using the methodology we propose the suitable solution can be found for each of them, by replacing the evaluation metric depicted in Equation 3.1 with the metric representative for the application-specific scenario and performing the rest of the analysis in the same way. In addition to balance between detection accuracy, detection time and power consumption being different for various application-specific domains, also the selection of the most representative metric varies based on the specific characteristics of the domain of interest, as described in Antunes and Vieira [2015], where a set of guidance on the selection of the most suitable metric is outlined. While this is a relevant research problem, it is not the focus of this thesis and not the main goal of the introduced metric. Namely, the main intention behind it in our scenario was to show how it would be used, as a part of the proposed methodology, in the application-specific domain once the suitable metric is known, rather than proposing its exact usage across different domains. Besides usage of the proposed detection systems with predetermined detection parameters, they can also be used in a form of an adaptive system, where based on the system dynamics the preferred parameters are

decided and changed during runtime executions.

3.3.6 Computational overhead

Although in the experimental setup we focus on measuring the average power consumed using our detection algorithms, the approach we propose to detect malicious applications takes into account limited resources of battery-operated devices in general. The algorithm used for detection of malware-related execution records, the reduced number of features considered, and the algorithm used for classifying malicious applications, are all compatible with these conditions. In fact, the algorithm chosen for malicious records classification has linear complexity in the number of features; for the algorithm used for malicious applications classification, very limited memory is required: for each running application, only the latest n record classification results, where n is the window length, need to be saved along with a counter of the number of past windows that have been marked as malicious. Only simple operations are performed and the complexity is linear with the window length. Both the number of features and the window length are chosen to be small.

3.4 Malware Classification

As it can be seen in Figure 3.1, in order to achieve the goal of discrimination between malicious records belonging to diverse malicious behavior, thus malware classification, we extend the approach depicted in Figure 3.4, so that, apart from being able to detect malicious records, is also able to differentiate between different malware families. Namely, as depicted in Figure 3.1, at runtime, based on the classification of execution records, malicious applications are detected and an alarm is raised. Then, recorded indicative features are sent into network for further analysis where malicious records classification is performed by using an ensemble of classifiers, including a classifier for each malware family. The development of the ensemble of detection methods is performed offline, as shown in Figure 3.3.

In order to identify to which malware family the running applications belong, we design a model that, based on the observed behavior, identifies the most indicative features for each malware family taken into account. We call this set of features *behavioral signature of a family*. Later, while looking at the behavior of the system and comparing it with the stored behavioral patterns, we can infer about the family that is being executed. It is our belief that creating be-

havioral signatures in this way, rather then typical signatures of malicious code samples, makes the detection system more resistant, because even if a sample is re-packaged, as long as it exposes the same behavior, it will be identified with the current signature, while in the other case a new signature would be required.

For this purpose, we observe features related to memory and CPU, and we also take into account network behavior of the considered applications. Network observation is included having in mind how different malware families interact with network in different ways (some connect to an external server to obtain further instructions, some send users sensitive data, etc) and assuming that this information might be significant in discriminating them.

Also in this scenario, in order to remove irrelevant features, to decrease complexity of monitoring infrastructure and required memory, we use feature selection as a separate step. In order to evaluate the importance of selected features and find out whether they contain enough information to discriminate between different families, we validate the ones selected as the most indicative by using the Support Vector Machines (SVM) model, that is a supervised learning model proven to be successful in many real world problems, that given a set of training examples, each marked for belonging to one of two categories (malicious or benign), builds a model that assigns new examples into one category or the other.

For the malware classification task, as aforementioned, we propose to offload into the cloud the computation related to the discrimination between different malware families. While in this way we decrease the computation load to be run on the phone and we allow usage of more sophisticated detection methods to identify the intention of running malware. We are aware that such approach rises other potential problems, the main of them being related to the privacy of users data. However, we believe that with the proposed method the privacy of the users is not significantly endangered due to the fact that the data is sent into a network only upon a detection of a possible attack and not constantly. In addition, the implementation of this part of the methodology in the real-time scenario would take into consideration the use of privacy-preserving methods in order to further protect the users private data.

Based on the type of malware being identified, we propose to apply suitable countermeasures. For example, the following ones can be considered:

- Malware that sends data over the Internet: new rules may be inserted in the mobile device local firewall to prevent the application from sending data; on Android devices, firewall is provided by Iptables [1] and rules can

[1] `https://www.netfilter.org`

be changed dynamically; the application can be optionally terminated.

- Malware applications sending SMSs may be terminated and their permissions may be changed so that, if the user starts them again, they are unable to access SMS.

Additionally, SELinux discussed by Smalley et al. [2001], that is included by default in Android since release 5.0, can be used to sandbox applications as also suggested by Shabtai et al. [2010]. While it might not be possible to do it while the applications are running, malicious applications can be terminated and the new SELinux policy can be applied when they are restarted.

In this section, we discussed the proposed methodology for the malware classification task. One of the important aspects of the proposed methodology is the selection of the suitable countermeasures and their communication to the device under attack. With respect to this, we outlined a set of possible countermeasures that could be suitable in the scenario of mobile malware detection. While detailed analysis of possible countermeasures is also an interesting research problem, the main focus, in this section, was rather on the introduction of proposed methodology for malware classification, the discussion on its main components and their main purpose from more generic point of view. Further discussion on detection of malware families and the initial results obtained using the proposed method are given in Section 5.3. The more detailed analysis of the malware classification related task and more precise response on the most suitable countermeasures is left for the future work, as it is discussed in Section 6.2.2.

3.5 Hybrid Method

The method we propose is based on dynamic features and their observation in order to discriminate malicious from benign applications. It is trained to detect applications at runtime, so that even if the applications expose their behavior later in the usage (for example in a scenario when the malicious payload is downloaded after the application installation), it could catch it, as opposed to static detection. However, in addition to the standalone usage of our method on mobile devices where the applications are not previously analyzed by any static approach, we proposed its usage also in a hybrid scenario, where the applications are first tested by a static detector based on the analysis of opcode 2-grams and then run on a device. For this purpose, we propose and evaluate a hybrid detection method, where static analysis is performed when applications are installed or updated and/or periodically (e.g., once a week), and dynamic

detection, instead, is used at runtime while the applications are executed and in a way described in Section 3.3. We depict the high-level workflow of the proposed approach in Figure 3.6: On those applications that were not marked as malicious by static detection and, therefore, allowed to execute, we use dynamic detection at runtime, while applications are executed. The main idea behind the proposed hybrid approach is to have the advantages of both static and dynamic approaches and to reduce their disadvantages, discussed in Chapter 2.

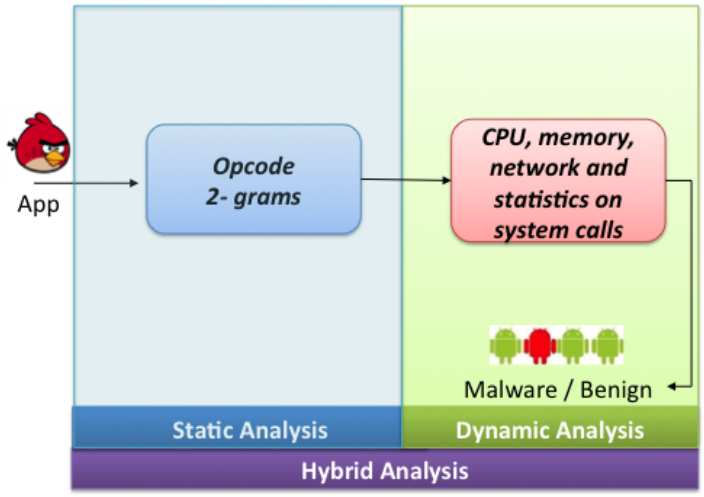

Figure 3.6. High-level workflow of the proposed hybrid approach. The static analysis consists of a classification using features obtained by the executable of the application under analysis, while the dynamic one by a feature set obtained when the application is running.

3.5.1 Static Analysis

We adopt the approach from Canfora, De Lorenzo, Medvet, Mercaldo and Visaggio [2015] for the static part of the proposed hybrid approach, due to its effectiveness in detection of Android malware and reported high detection rate. In this approach each application is pre-processed in order to obtain the numeric values of frequencies of opcode sequences that are suitable to be processed by the classifier. After pre-processing, the classifier undergoes the learning phase in which it is trained by using a labelled dataset. After the learning phase, the classifier can be used for the actual classification of the applications as malware or benign.

Dalvik virtual machine at runtime is able to run dalvik executables (.dex files). However, for humans .dex files are unreadable, and to edit them we need to convert this .dex files to a more understandable form, for which smali is used is this scenario. Namely, converting a .dex file to smali (that is called baksmaling) gives us readable code in smali language, that can be then modified and converted back in .dex format. Also, from java source code we can create .smali representation of it. In the considered scenario, if a is the Android application under analysis, in the pre-processing phase of this method, the apktool [2] tool is first used in order to extract from the `.apk` the Smali classes of the application under analysis in form of `.dex` files; then, from these files a unique file containing the full set of opcodes (without the relative argument and parameters) relative to all the a application classes is obtained.

The frequency of 2-grams opcodes from the previously obtained unique file is computed as follows: let O be the set of possible opcodes, and let $\mathcal{O} = \bigcup_{i=1}^{i=n} O^i$ the set of n-grams. If $f(a,o)$ is the frequency of the n-gram $o \in \mathcal{O}$ in the application a: $f(a,o)$ is the number of occurrences of o divided by the total length of the opcode sequences in a. Finally, the *feature vector* $\vec{f}(a) \in [0,1]^{|\mathcal{O}|}$ corresponding to a to $\vec{f}(a) = (f(a,o_1), f(a,o_2), \dots)$ with $o_i \in \mathcal{O}$ is set. In order to consider n-grams of the same method, the application code is split into chunks corresponding to class methods.

In the next phase, named learning, a binary classifier C from two sets A_M, A_T of malware and benign applications (the *learning sets*) is trained, respectively. The learning phase is divided into a feature selection phase and the actual classifier training phase. As previously stated, the considered n-grams is $n = 2$ because a previous work of Canfora, De Lorenzo, Medvet, Mercaldo and Visaggio [2015] demonstrated that the sequences of two consecutive opcodes obtain better performance in identifying Android malware than with $n = 1$ and with other values of n ranging from 3 to 5. At first the average frequencies $\bar{f}_M(o)$ and $\bar{f}_T(o)$ are computed for each 2-gram $o \in \mathcal{O}$ on the malware and benign samples, as follows:

$$\bar{f}_M(o) = \frac{1}{|A_M|} \sum_{a \in A_M} f(a,o)$$

$$\bar{f}_T(o) = \frac{1}{|A_T|} \sum_{a \in A_T} f(a,o)$$

Then, the relative difference $d(o)$ between the two average values is computed

[2] `https://ibotpeaches.github.io/Apktool`

as follows:

$$d(o) = \frac{\text{abs}(\bar{f}_M(o) - \bar{f}_T(o))}{\max(\bar{f}_M(o), \bar{f}_T(o))}$$

This relative difference is high if the 2-gram o is frequent among malware applications and infrequent among benign applications (and vice versa). Then, the set $\mathcal{O}' \subset \mathcal{O}$ of n-grams composed of the h n-grams with the highest values of $d(o)$ is build, where h is a parameter of the method proposed by authors. The 2-grams that appear only in benign or malware applications are excluded, so as the 2-grams that are redundant, and the *reduced feature vector* $\vec{f}'(a)$ is set corresponding to a using only the frequencies of the 2-grams in $\mathcal{O}'$, i.e., $\vec{f}'(a) = (f(a, o_1), f(a, o_2), \dots)$ with $o_i \in \mathcal{O}'$. The second step of the learning phase consists of training the actual classifier C using the reduced feature vectors obtained from the applications in the learning sets and the corresponding labels, for which the following classification algorithms are used: J48, NaiveBayes, and LR.

In the last phase, named classification, according to the learned classifier C is determined if an application a is labelled as malware or benign. To this end, a is pre-processed as previously explained in order to obtain the reduced feature vector $\vec{f}'(a)$. Then, this vector, $\vec{f}'(a)$, is used as an input to C and a is classified into $\{\text{malware}, \text{benign}\}$. In the envisioned hybrid scenario, this step is run every time a new application is installed or updated.

3.5.2 Dynamic Analysis

While for the static part of the hybrid approach, we use the described existing model in the state of the art, for the dynamic part, we use our approach that is based on a two-steps detection system of low complexity that first classifies execution records, and then complete applications by relying on the past classifications of execution record, that we previously introduced in Section 3.3.

The proposed hybrid approach is evaluated using ransomware malicious applications, since ransomware is type of malware that is currently one of the most serious threats to security of mobile devices and is particularly on the rise in the last two years, as stated in Section 2.1. The results obtained using this approach on a representative ransomware and benign applications dataset are discussed in Section 5.4.

3.6 Key Properties of the Proposed Approach

In this section we discuss the main properties of the proposed approach. Having in mind previously introduced overall methodology described in Section 3.1, the selection of the indicative features outlined in Section 3.2, the proposed methodology for malware detection described in Section 3.3, the methodology aimed for malware classification introduced in Section 3.4, and the hybrid scenario in which the proposed method can be used presented in Section 3.5, we can summarize the key properties as follows:

- It provides protection against malicious infections, at low cost, in resource-constrained environment by maximizing F-measure with respect to detection time and power consumption
- It is based on the observation of the dynamic aspects of the system behavior and as such is able to detect previously unseen malware samples and is more resistant against obfuscation than commonly used static methods
- It is adaptive and can be suited for a variety of resource-constrained devices, since the number of observed features can be changed with respect to the environment in which the system is used and the observed metric can be optimized with the applications-specific conditions
- It is used at runtime, when it selects and monitors only the most indicative features able to discriminate between malicious and benign behavior
- It uses customized detection algorithms of low complexity to perform on device malware detection
- It is both effective and efficient and provides early detection of mobile malware
- It is able to detect both parts of the malicious applications executions and the complete malicious applications
- It can distinguish between diverse malicious behavior of running applications
- It can be used as a standalone approach or in a combination with static analysis, as a part of a hybrid approach

- It is data-driven, where parameters are decided based on the experimental results based on the observation of the influence of behavior of both malicious and benign applications on the system parameters
- It increases systems security and by early detection of attacks also increases systems uptime thus contributing to the overall increase of systems dependability
- It is demonstratable on the high-impact operating system such as Android OS.

Chapter 4

Setup for the Experiments

The approach we proposed and discussed in Chapter 3 can be applied to any operating system. However, since such approach is the most suitable for the battery-operated devices on one side, and on the other, according to *Cisco 2014 Annual Security Report* [2014], the vast majority of known mobile malware targets Android devices, and, there is large availability of malware samples for mobile systems based on Android OS, we use both malicious and benign applications running on it and its emulation environment in our study. Additionally, Android OS has by far the largest install base worldwide and provides developers and researchers with sufficient freedom to carry out the foreseen tasks. In the remaining part of this chapter, we first discuss the dataset and the execution environment we used in order to test our approach. Later, we describe the features that we collected and introduce the feature selection methods used in order to reduce the number of monitored features and identify only the most indicative ones for both malware detection and malware classification tasks. Finally, we give details on the environment that we used for the power consumption evaluation of the developed detection methods and the environment we used for development and testing of the proposed methodology.

4.1 Datasets

One important part of the proposed methodology is the selection of a suitable dataset of representative applications to be executed. Namely, on one side it is important to use malware samples originating from different malware families, so that the diverse behavior of malware is covered to as large extent as possible, and on the other a set of benign applications coming from various available categories. In order to ensure it, we used a dataset consisting of a variety of malware

families with broad scope of behavior. Furthermore, it is also needed to set up the execution environment to run malware samples, so that malicious behavior is activated and with the possibility to execute large number of samples at the same time, so that the obtained results have statistical significance. In order to achieve these we used an emulated environment in which we set up an automatic method for triggering different events while executing applications.

Since our goal is to discriminate between malware and benign applications, we have taken into account also benign samples, covering a wide range of application categories, and executing them in the same conditions used for malicious applications.

The datasets that we use in the experiments consist of a mix of benign and malicious applications, that we describe in the remaining of this section.

4.1.1 Benign applications collection

With the widespread presence of malware, greyware and madware, as introduced in Section 2.1, some of it being hidden within the applications without exposing its intents until certain conditions are met, it is difficult to guarantee that the collected applications are absolutely benign. However, in order to be more confident about it, we have selected to download all the applications from the official *Google Play Store* [2015], where they are investigated by Google for potential malicious content. Additionally, to further increase the probability of having malware-free applications, the benign applications retrieved are among the most frequently downloaded ones in their category, and they are further analyzed with the Virus Total [n.d.] service, that runs 57 different antimalware software on each submitted application and its output confirmed that the benign applications did not contain any malware.

In order to cover a variety of benign behavior of Android applications, the downloaded applications belong to a set of different categories, listed in Table 4.1.

4.1.2 Malicious applications collection

Up to date, two datasets with malicious applications for Android OS were released for the research community: Malware Genome, released by Zhou and Jiang [2012] and Drebin dataset, made public by Arp, Spreitzenbarth, Huebner, Gascon and Rieck [2014] and by Spreitzenbarth, Echtler, Schreck, Freling and Hoffmann [2013]. We collected malicious applications from both of them and we used them in the evaluation of our approach.

Table 4.1. Considered categories for analyzed benign applications taken from GooglePlay.

Books & Reference	Lifestyle	Business	Live Wallpaper
Comics	Media & Video	Communication	Medical
Education	Music & Audio	Finance	News & Magazines
Games	Personalization	Health & Fitness	Photography
Libraries & Demo	Productivity	Shopping	Social
Sport	Tools	Travel & Local	Transportation
Weather	Widgets		

Malware Genome dataset is released in year 2012 as a part of Malware Genome project, with a goal to mitigate malware threats on Android mobile platform and engage the research community to better understanding and defense. The authors collected the dataset by crawling both manually and in the automated way a variety of Android Markets. The dataset is composed of 1,260 Android malware samples, that cover the variety of malicious behavior belonging to 49 different malware families. The detailed analysis of this dataset, performed by Zhou and Jiang [2012], showed that the released malware families cover a broad range of existing malware behavior, including different installation methods (repackaging, update, drive-by download, standalone), activation mechanisms (Boot Completed, Phone Events, SMS, CALL, USB, Battery, Package, Network, System Events) and different malicious payloads (privilege escalation, remote control, financial charges, and personal information stealing). According to the authors, 1,083 out of the considered applications (thus 86%) are repackaged versions of legitimate applications with malicious payloads; one third of the applications (36.7%) uses some root-level exploits to compromise the system; more than 90% of the applications turn the compromised device into a botnet controlled through network or short messages; among the considered families, 28 of them have built-in capabilities to send messages to premium-rate numbers or to make calls without users awareness; 27 families are harvesting information about users, together with users accounts and users messages.

To foster research in Android malware detection and enable comparison of performance of different approaches the authors of Drebin, released their dataset too. This dataset, introduced and discussed by Arp, Spreitzenbarth, Huebner, Gascon and Rieck [2014] and by Spreitzenbarth, Echtler, Schreck, Freling and Hoffmann [2013] consists of 5,560 applications from 179 different malware families. The samples have been collected in the period of August 2010 to October 2012 and were made available by the MobileSandbox project, that is a web-based tool that takes an Android application (apk-file) and analyzes it for malicious be-

haviour. The tool is proposed and described by Spreitzenbarth, Echtler, Schreck, Freling and Hoffmann [2013].

4.1.3 Dataset used in malware detection task

The experiments with the malware detection task were performed using Malware Genome dataset and 950 benign applications coming from the Play Store, as outlined in Table 4.2. Based on this dataset, we evaluated the detection accuracy of the malicious records detection and malicious application detection systems that we designed. Additionally, using it, we also experimented with the most suitable sampling period.

Table 4.2. Malware detection dataset description

Category	Number of families	Number of Samples
Malware Applications	41	1120
Benign Applications	–	950

4.1.4 Dataset used in malware classification task

For the malware classification task, the used malware and benign applications, together with their number are shown in Table 4.3. The five Trojan families, taken from Malware Genome and Drebin dataset, that were considered as a proof-of-concept, are following:

- **DroidKungFu:** Once installed, Trojans in this family attempt to gain control of the system by using exploits that are stored in the malware package and encrypted with a key. DroidKungFu collects the following pieces of information from the phone: International Mobile Equipment Identity (IMEI), Mobile device model, network operator and type, OS APIs and type, and information stored in phone and SD card memory.

- **Fake Player:** Trojans belonging to this family pretend to be a movie player, but instead send SMS messages.

- **Geinimi:** Trojans belonging to this family send personal data (location coordinates, device identifiers, the list of installed apps) to remote servers.

- **GinMaster:** If triggered, Trojans in this family harvest confidential information from devices without users knowledge nor consent. Phone identification numbers (IMEI and IMSI, SIM number, telephone number), network type, current version of applications, and serial number are stolen by this malware in this family.
- **Kmin:** Trojans in this family collect user and device data (Device ID, Subscriber ID, current time) and send it to a remote server.

Table 4.3. Malware Classification dataset description

Category	Name	Number of Samples
Malware	DroidKungFu	667
	Fake Player	6
	Geinimi	92
	Ginger Master	339
	Kmin	147
Benign Apps	–	300

4.1.5 Dataset used in the evaluation of the hybrid method

Dataset used in the evaluation of the hybrid method contains 3,058 mobile applications: 2,386 Android benign applications downloaded from the Google Play Store and 672 applications containing malware of type ransomware taken from the freely available HelDroid dataset[1], as outlined in Table 4.4. These malicious samples appeared form December 2014 to June 2015. The list of malware families to which these samples belong, as introduced by Andronio et al. [2015], follows:

- Malicious applications in the *Locker* family are able to block the screen of an infected device and requests a ransom for unlocking the device. No files are actually encrypted.
- The *Koler* payload is delivered with site redirection; once installed and run the screen is taken over by the ransom browser page; pressing the Home button or attempting to dismiss the page works for a very short time as the page reappears when users attempt to open another application or within a few seconds.

[1] `http://ransom.mobi`

- Malware in the *Svpeng* family are based on an overlay attack: legitimate applications launched by the user are overlayed with a fake window imitating the legitimate applications and thus fooling the victim. Additionally, users receive a message, which claims to have been sent by the FBI, explaining that it has been used to access child pornography sites and has been locked until fine is paid via MoneyPak.
- Samples in the *ScarePakage* malware family masquerade as well-known applications, such as Adobe Flash or antimalware applications; and, when launched, they pretend to scan your phone. After completing the fake scan, the device is locked and after a reboot a fake FBI message is shown. A ransom of several hundred dollars in a MoneyPak voucher is asked to bring back the device to normal. The application does not need root permissions in order to take over the phone, but it does need device administrator privileges.
- Malicious applications in the *SimpleLocker* family scan the SD card for images, documents and videos and encrypt them by using AES; a message notifying the user and asking for a ransom is shown on the display. This is the only family in our dataset that actually encrypts data on the device and it was the first one discovered for Android.

Table 4.4. Hybrid approach dataset description

Category	**Number of families**	**Number of Samples**
Malicious Ransomware Applications	5	672
Benign Applications	–	2386

For development and validation of the proposed approach, we have analyzed the collected data both by separating the acquired applications into a training, test and validation datasets and by using cross-validation on the complete dataset.

4.2 Database creation

The database we create consists of the execution records of both malicious and benign applications. Features of interest have been recorded by running the applications, one at a time, on the Android emulator and by using a script that recorded their usage periodically; each application has been run for 10 minutes.

Although it could be the case that a longer execution period would provide more significant results, we believe that the duration we have chosen is a good trade-off between time when most of the malware samples expose their malicious intentions and duration of the overall experimentation. This claim is additionally supported by the results on detection time discussed in Section 5.2.4.

In order to guarantee that the executed malicious samples did not cause any external damage, we have used an emulated environment for the execution of the applications. This was done for both benign and malicious applications in order to have a database with consistent execution traces. The Android emulator of choice in our case is the one included in the Android Software Development Kit released by Android Open Source project [2015b] under number 20140702, running Android 4.0 (Ice Cream Sandwich). Additional reason why an Android emulator has been chosen instead of real devices is that this solution provides the ability to run a large number of applications, making the obtained dataset more significant. The Android OS has been each time re-initialized before running each application. In this way, we were able to avoid possible interference from previously run samples (e.g., changed settings, running processes, modifications of the operating system files). In the Android 4.0 version that we use in order to perform the experimental evaluation, the Android uses Dalvik Just in Time (JIT) compiler, in order to have smaller footprint and use less space on the device, that each time when the app is run, it dynamically translates a part of the Dalvik bytecode (.dex) into native machine code.

The procedure of executing the applications has been automated by means of a Linux shell script, which has been run on a Linux PC and made use of Android Debug Bridge (adb) introduced by Android Open Source project [2015a], a command line tool that allows the PC to communicate with an emulator instance or with an Android device. The Monkey application exerciser released by Android Open Source project [2015c] has also been used in the script. The Monkey is a command-line tool that can be run on any emulator instance or on a device; it sends a pseudo-random stream of user events into the system, which acts as a stress test on the application software.

In our script, Monkey has been used to activate different parts of the applications; adb has been used to monitor application features, as well as to install the applications. In summary, the following actions have been performed for each application:

- A clean installation of Android OS is performed on the emulator
- The application is installed by means of adb

- The monitoring of dynamic features of interest is initialized
- The application is started and run for 10 minutes by using Monkey.

Due to the type of features considered, limited or no influence is expected from the fact that the emulator is used instead of real devices.

The database contains the execution traces of the collected malicious and benign applications and their influence on the observed dynamic parameters of the system. In order to make the database as representative as possible we collected and executed those malicious samples that are released publicly for research purposes. Although we are aware that there is a variety of other malware samples present on the market, due to the fact that they are not publicly released we could not take them into account, which makes our collected database more suitable for research rather than industrial purpose. However, to the best of our knowledge, this is the first database of this kind and we believe that it can be used in order to facilitate the research in the area of dynamic malware detection methods and their further adoption in real-time scenarios.

4.3 Collected features

We have collected dynamic features related to memory, CPU and network behavior, and statistics on system calls as well as generated system calls of the observed applications executions. The collected features together with categories they belong to, are listed in Table 4.5.

4.3.1 Memory and CPU related features

We have taken into account all the features related to memory and CPU that could be accessed in Android OS from adb; in total, we have recorded 53 features. Features are extracted and composed in execution records (i.e., feature vectors) that represent the behavior of the system with a 2s sampling period.

In detail, we have extracted three CPU-related features related to single applications under analysis:

- Total: total percentage of CPU used
- User: percentage of CPU used by user
- Kernel: percentage of CPU used by kernel

Memory related features are obtained by monitoring the following types of memory consumption: (i) the Dalvik Virtual Machine; (ii) the native memory usage; (iii) the total memory usage. In particular, for each of these three categories, we have extracted the following features:

- PSS: The total Proportional Set Size (PSS) is the RAM used by proces; it indicates the overall memory weight of a process, which can be directly compared with other processes and the total available RAM
- Shared: Shared Dirty is the amount of shared RAM that will be released back to the system when the process is destroyed; Dirty RAM is represented by pages that have been modified and must stay committed to RAM, since there is no swap in Android OS
- Private: Private Dirty is the amount of RAM that will be released back to the system when the process is destroyed
- Heap allocation: It represents the RAM actually allocated by Dalvik Virtual Machine for the application under analysis
- Heap free: Represents the allocatable RAM by Dalvik Virtual Machine for the application under analysis

All considered memory features are about single applications under analysis.

4.3.2 Network behavior related features

Network statistics have been obtained by logging all network traffic of the emulator and by successively running the *tcpstat* tool introduced by Herman [2009], set to consider 2s sampling period. The network features contain network statistics at transport, link and Internet layer. The collected information about the network traffic is grouped into the network features depicted in Table 4.5.

4.3.3 Observed system calls and features related to the statistics of system calls

To collect system calls we used strace [2], a tool for tracing system calls. In particular, we used the command `strace -p PID` in order to hook the process corresponding to the application under analysis and intercept only its system calls.

[2] `http://linux.die.net/man/1/strace`

Table 4.5. List of all the considered features; totals are related only to single applications; unless differently specified, all numbers are related to the considered sample rate of 2s.

Category		Feature Names
CPU	CPU Usage	Total CPU Usage, User CPU Usage, Kernel CPU Usage
	Virtual Memory	Page Minor Faults, Page Major Faults
Memory	Native memory	Native Pss, Native Shared Dirty, Native Private Dirty, Native Heap Size, Native Heap Alloc, Native Heap Free
	Dalvik memory	Dalvik Pss, Dalvik Shared Dirty, Dalvik Private Dirty, Dalvik Heap Size, Dalvik Heap Alloc, Dalvik Heap Free, Cursor Pss
	Cursor memory	Cursor Shared Dirty, Cursor Private Dirty
	Android shared memory	Ashmem Pss, Ashmem Shared Dirty, Ashmem Private Dirty
	Memory-mapped native code	.so mmap Pss, .so mmap Shared Dirty, .so mmap Private Dirty
	Memory mapped Dalvik code	.dex mmap Pss, .dex mmap Shared Dirty, .dex mmap Private Dirty
	Memory-mapped fonts	.ttf mmap Pss, .ttf mmap Shared Dirt[ ]y, .ttf mmap Private Dirty
	Other memory-mapped files and devices	.jar mmap Pss, .jar mmap Shared Dirty, .jar mmap Private Dirty, .apk mmap Pss, .apk mmap Shared Dirty, .apk mmap Private Dirty, Other mmap Pss, Other mmap Shared Dirty, Other mmap Private Dirty
	Non-classified memory allocations	Unknown Pss, Unknown Shared Dirty, Unknown Private Dirty, Other dev Pss, Other dev Shared Dirty, Other dev Private Dirty
	Memory Totals	TOTAL Pss, TOTAL Shared Dirty, TOTAL Private Dirty, TOTAL Heap Size, TOTAL Heap Alloc, TOTAL Heap Free
	Objects	Views, ViewRootImpl, AppContexts, Activities, Assets, AssetManagers, Local Binders, Proxy Binders, Death Recipients, OpenSSL Sockets
	SQL	heap, MEMORY_USED, PAGECACHE_OVERFLOW, MALLOC_SIZE
Network	Link layer networking	Number of ARP packets, AVG. PKT Size bytes, bps, Number of ICMP packets, Size in byte standard deviation
	Internet layer networking	Number of IPv4 packets, Network load over last minute, Maximum packet size in bytes, Minimum packet size in bytes, Number of bytes, Number of packets, Number of packets per second, Number of IPv6 packets
	Transport layer networking	Number of TCP packets, Number of UDP packets
System calls		Number of Syscalls, No. of different syscalls, Average no. of calls per syscall, No. of calls occurring once, No. of calls occurring multiple times

Apart from collecting system calls during the execution of the applications, we have also taken into account, as possibly indicative features, following statistics on collected system calls: number of system calls, number of different system calls, average number of calls per system call, number of calls occurring once, and number of calls occurring multiple times, as depicted in Figure 4.5.

4.4 Features Selection

Although, intuitively, the higher the number of features the better classification results should be, in practice this is not always the case as it can happen that irrelevant and redundant features confuse classifiers and decrease detection performance. Due to this fact, we have performed feature selection as a separate

step, in which we have evaluated the importance of all previously introduced collected features. An extensive survey providing more insight about the usefulness of feature selection methods and covering the state of the art in feature selection is proposed by Liu and Yu [2005].

The main purpose of the feature selection step is to find the features subset that represents the best the original set of features with respect to some predefined criteria. The existing methods can be divided into filter, wrapper or hybrid. Filter based feature selection models evaluate the importance of features independently from any specific classifier, using criteria based on independent measures (i.e., information gain). Wrapper methods are focused on the selection of the features subset that is the most representative for the selected classifier of choice. Hybrid methods use combination of filter and wrapper approach.

In this thesis, we focus on the evaluation of the importance of features using filter methods, and thus measure of the features importance independent from the used set of classifier. The main motivation behind this choice is to keep the proposed methodology modular, so that, in example, if we have to replace a classification module, we do not necessary have to replace the features monitoring part at the same time.

In order to find which features contain the highest amount of information about the behavior of the running applications, we have evaluated their importance from different perspectives, namely different independent measures. With respect to this, we have taken into account features selection filter based methods that observe usefulness of features based on statistical importance or information gain measure or correlation measure between attributes. More precisely, we have evaluated features usefulness based on the following techniques: Principal Component Analysis (PCA), Correlation Attribute Evaluator, CFS Subset Evaluator, Gain Ratio Attribute Evaluator, Information Gain Attribute Evaluator, and OneR Feature Evaluator.

Following is the short description of the feature selection algorithms that we have used:

- *PCA* is a method that is able to identify how different variables work together to create dynamics of the system. Additionally, it can reduce the dimensionality and decrease redundancy in the data, filter the noise, and prepare the data for further analysis using other techniques, as stated by Shlens [2005]. PCA method is already proven in the literature as a successful method in the selection of indicative features for detection of PC malware, as pointed out by Yen and Reiter [2008], by Vinod et al. [2012], and by Nair et al. [2010].

- *Correlation Attribute Evaluator* calculates the worth of an attribute by measuring the correlation between it and the class. An attribute X is preferred to another attribute Y if the correlation between attribute X and class C is higher than the correlation between Y and C.

- *CFS Subset Evaluator* calculates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them. Subsets of features that are highly correlated with the class while having low intercorrelation are preferred, as discussed by Hall [1998].

- *Information Gain Attribute Evaluator* calculates the worth of an attribute by measuring the information gain with respect to the class. The information gain from an attribute X is defined as the difference between the prior uncertainty and expected posterior uncertainty using X. In this context, attribute X is preferred to attribute Y if the information gain from X is greater than that from Y.

- *Gain Ratio Attribute Evaluator* calculates the worth of an attribute by measuring its gain ratio with respect to the class. The information gain ratio is a ratio of information gain to the intrinsic information. It is used to reduce a bias towards multi-valued attributes by taking the number and size of branches into account when choosing an attribute [3].

- *OneR Feature Evaluation* calculates the worth of an attribute by using the OneR classifier, which uses the minimum-error attribute for prediction. Although simple, this method is shown to perform well in many practical scenarios, as pointed out by Holte [1993].

For the mentioned feature selection methods their implementations in the Weka tool, proposed by Hall et al. [2009], has been used. Weka is a data mining tool based on Java that we connected with our experimental environment and that we used due to already proven bug-free algorithms implementations.

4.5 Power Consumption Measurement

We implemented the detection system we propose as an Android application, that we have later used to measure its power consumption. The developed application

[3] `http://www.ke.tu-darmstadt.de/lehre/archiv/ws0809/mldm/dt.pdf`

was tested on a Google Nexus 5 mobile phone [4]. The CPU power consumption was evaluated by using PowerTutor, an application that displays the power consumed by major system components such as CPU, network interface, display, and GPS receiver introduced by Zhang et al. [2010]. PowerTutor's power model was originally built on HTC G1, HTC G2 and Nexus One. On other Google phones, like ours, it provides rough estimation of the power. Although power estimation is approximate, we still find it suitable for relative comparison between power consumption of our detection system when different sampling periods are employed, the main purpose we use it for. Namely, the way we collect consumed battery power measures is that we run on the phone the implemented application and we observe the consumed power while changing sample rates of interest (2s, 4s, 6s, 8s, 10s, 12s, and 16s). For each of the sampling periods we observe the battery power consumption for one hour and we repeat the same experiment three times, after which we average the reported battery power consumption, and we use it for the evaluation of the power consumption of the observed detection solution. Between each of the measures the phone is restarted, in order to guarantee that there is no dependence between current and previously executed experiment. Since from the point of view of the developed detection system the complexity remains the same in case of malicious or benign executions, in order not to cause any damage to the used devices, the power consumption of the developed method was evaluated while only benign applications were running on it.

4.6 Development Environment

In addition to the automated process of the applications executions, introduced in Section 4.2, we have also automated the process of features selection and detection algorithms training. The main development environment used for these tasks is Java based Intellij Idea [5], Java integrated development environment, that is connected with Weka tool, from which different feature selection methods and different machine learning classifiers were used. The proposed method is implemented as an application for Android OS, using Android Studio [6], that is the official integrated development environment for Android.

[4] https://www.androidcentral.com/nexus-5-specs
[5] https://www.jetbrains.com/idea/
[6] https://developer.android.com/studio/index.html

Chapter 5

Experiments and Results

With increased use of mobile devices also the interest of attackers to abuse them is increasing, which further results in the increased number of malware samples and malware families, that attackers develop in order to carry out their malicious intents. In Section 2 we investigate existing mobile threats and proposed detection solutions. Our investigation has shown that most of the currently proposed methods are either too complex to be employed on battery-operated mobile devices or provide insufficient detection accuracy against a wide range of malware and against unknown malicious samples. Additionally, most of the current solutions are not able to detect parts of malicious applications executions nor to identify malicious families being executed. Finally, they frequently do not take into account detection time, so it is not clear if the system would be able to respond in time and actually protect the user against malicious infection. Due to this, we focused our research efforts on runtime detection of malicious parts of applications, complete malicious applications, and discrimination between different malware families, thus malware classification, while having in mind the resource-constrained environment in which these algorithms have to be used and while being focused on early detection of the malicious infections. In the remaining part of this chapter we discuss results we obtained performing this investigation.

One of the most important parts of the proposed methodology, introduced in Section 3, is the identification of the most indicative features to be monitored at runtime in order to detect and identify presence of malware. The features that we identified for this task we discuss in Section 5.1. Once the most indicative features are identified, a detection algorithm suitable for limited resources of mobile devices has to be used. Since our goal was to detect both parts of malicious application and complete applications, we developed two detection

methods: one for malicious records detection and the other for malicious applications detection. In addition to being able to understand which parts of the running applications are malicious, in this way we can also customize our detection system with respect to detection time and detection accuracy. Results we obtained in detection of the most indicative features for the discrimination between malware and benign applications are discussed in Section 5.2.1, followed by the results in the detection of malicious execution records that are presented in Section 5.2.2. Detection results on complete applications are given in Section 5.2.4. In Section 5.2.5we investigate the importance of the sampling period of the monitoring infrastructure for the efficient malware detection, and discuss dependencies between detection accuracy, detection time and consumed power with respect to it. Apart from designing a system being able to discriminate between malicious executions records and applications, we also propose a solution for detection of groups of malicious executions (sub-traces). The results we obtained in this domain are given in Section 5.2.3. Finally, after identifying that most of the current detection solutions are not able to provide more information about the nature of the executed program than it is malware or benign, we investigated and provided initial results in detection of parts of executions that are belonging to mobile malware families. These results are presented in Section 5.3. Additionally, in Section 5.3 we discuss the proposed unified malware detection and malware classification system, that performs malware detection on device and, in case malware is being detected, sends the most indicative features for the malware classification task into a network for the further analysis and the decision on countermeasures. The methods we propose for both malware detection and malware classification tasks are standalone, dynamic methods, intended for the protection of malware that appears at runtime. Apart from their usage as a standalone method, we also investigate the performance of a hybrid approach, combined static and dynamic detection, and in Section 5.4 we discuss the obtained results.

The experiments we performed in this work and the results we report in this chapter are with a set of mobile devices running applications on Android OS. However, due to the simplicity of monitoring and the low-complexity of algorithms used to detect malware, we are confident that our approach can also be used in majority of other IoT devices.

5.1 Identification of the most indicative features for efficient and effective malware detection and malware classification

In order to select the most indicative features that can be used to discriminate malicious from benign behavior, and to discriminate between different types of malicious behaviors, we investigated the importance of memory and CPU related features. On one side, these features are easily accessible on a phone which is in line with our goal to have a monitoring infrastructure of low complexity. On the other side they reflect well the behavior of the system.

We have executed malware samples using the experimental setup described in Chapter 4 and we have observed their influence on all memory and CPU features representing phone state that we could collect (53 features in sum, 48 for memory occupation and 5 for CPU usage, described in Section 4.3). We have analyzed the collected execution traces by using PCA, introduced in Section 4.4. Results have been obtained by analyzing following 25 malware families, taken from the Malware Genome project described in Section 4.1.2 and released by Zhou and Jiang [2012]: AnserverBot, Asroot, BaseBridge, BeanBot, CoinPirate, CruseWin, DroidDream, DroidDreamLight, DroidKungFu1, DroidKungFu2, DroidKungFu3, Fake Player, Geinimi, Ginger Master, Gold Dream, Hippo SMS, jSMSHider, KMin, Pjapps, Plankton, Rogue Lemon, Rogue SPPush, SndApps, zHash, and Zsone; since these families cover broad behavior of malware.

After applying PCA on the features collected during the execution of mobile malware samples, we ranked their importance within the families used. Then, we calculated the occurrence of each of the analyzed features among the top ranked 5 and top ranked 15 features. Top 5 and top 15 features are chosen to be presented, because it is a number of features that is compatible with the limited resources of a smartphone, both for monitoring and detection.

Information about the number of occurrences of features among top 5 the highest ranked is given in Figure 5.1 and the information about number of occurrences of features among top 15 the highest ranked is given in Figure 5.2. Although 53 features were used for the analyses, not all of them appeared as promising candidates for detection. Namely, only 15 of them appear among top 5 of the most indicative features (as it can be seen in Figure 5.1) and only 25 of them appear among top 15 indicative features (as it can be seen in Figure 5.2).

While Figures 5.1 and 5.2 illustrate which features are the most frequent, further insights into the importance of features can be observed by looking at their average position within rankings. This ranking is depicted in Figures 5.3

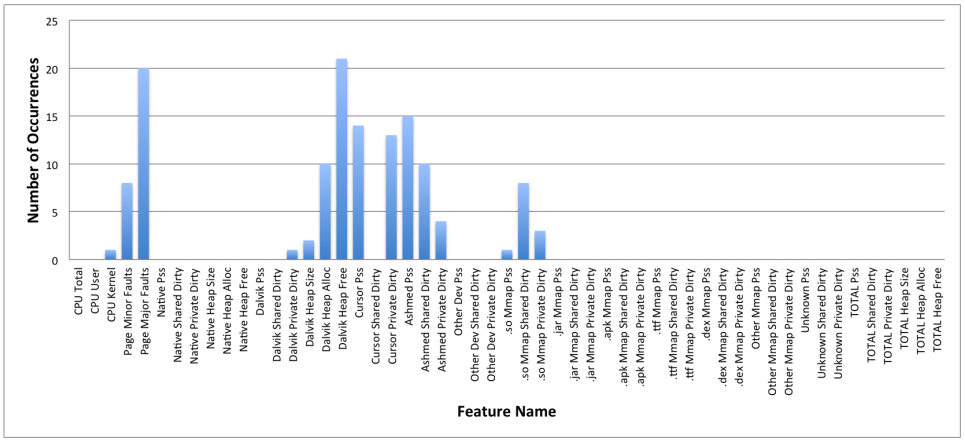

Figure 5.1. Number of occurrences of memory and CPU usage related features among the top 5 most indicative ones

and 5.4 that represent the average position of the 15 features shown in Figure 5.1 and 25 features shown in Figure 5.2. Together with average ranking, also the number of occurrences of the features is included in the figures.

The figure shows that there are some features, such as *Page Major Faults* and *Cursor PSS*, that, besides appearing very frequently, are also often ranked high. Conversely, features that are not appearing frequently, such as *CPU Kernel* and *Dalvik Private Dirty*, are ranked low.

Based on our studies we can observe the following: some features are not appearing among the indicative ones, thus being irrelevant for further investigation; some features appear as indicative for almost each of the investigated malware families; and some features appear to be indicative only for some of the analyzed malware families.

Features indicative overall are features that appear as indicative in almost all investigated malware families, having at the same time high average rank. We believe these features are good candidates for further investigation towards identification of malware in general, thus malware detection. According to Figure 5.3 such features are *Page Major Faults*, *Dalvik Heap Free*, *Ashmem PSS* and *Cursor PSS*. According to Figure 5.4 representative features are *Native Shared Dirty*, *Native Private Dirty*, *Dalvik Private Dirty*, *Dalvik Heap Size*, *Dalvik Heap Alloc*, and *Dalvik Heap Free*. These features represent different aspects of programs behavior. *Page Major Faults*, appear when the page is not loaded in memory at the time the fault is generated. *Dalvik Heap Free* is the free RAM that can be used by Dalvik allocations in the applications. Proportional Set Size (PSS) is the measurement of the applications usage that takes into account sharing pages across

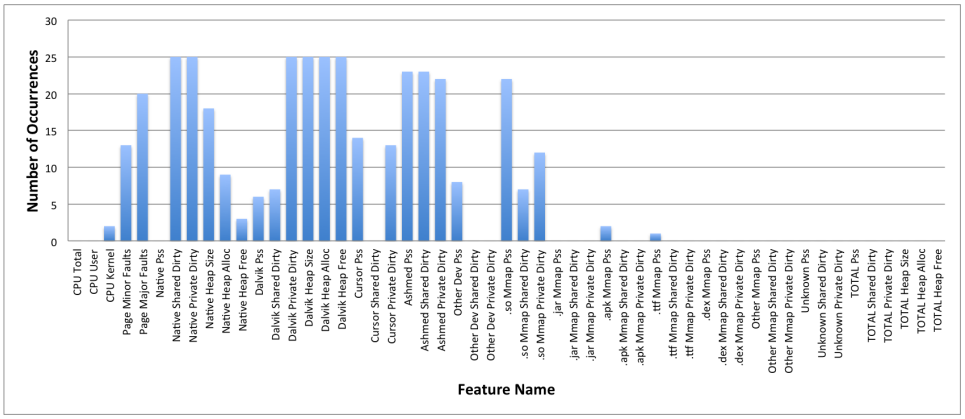

Figure 5.2. Number of occurrences of memory and CPU usage related features among the top 15 most indicative ones

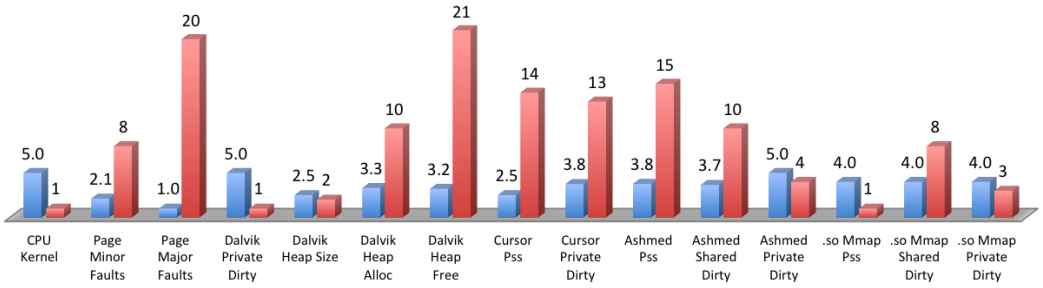

Figure 5.3. Average ranking of the features that appear among the top 5 the most indicative ones. Blue color depicts the average rank of the observed features, while red color depicts its number of occurrences.

processes. *Ashmem PSS* and *Ashmem Shared Dirty* are describing Android shared dynamic RAM usage across processes using explicitly allocated shared memory regions.

Features indicative for specific malware families are features that appear rarely as the most indicative, but when they appear they have high average rank. We believe that these features are good candidates for further investigation towards identification of specific malware families, thus malware classification. According to Figure 5.3 such features are *Dalvik Heap Size* and *Page Minor Faults*. According to Figure 5.4 relevant features are *Cursor PSS* and *Cursor Private Dirty*. More in detail, *Page Minor Faults* appear when the page is loaded into memory at the time the fault is generated, but it is not yet marked in the memory management unit as being loaded into memory. This could happen if the memory is

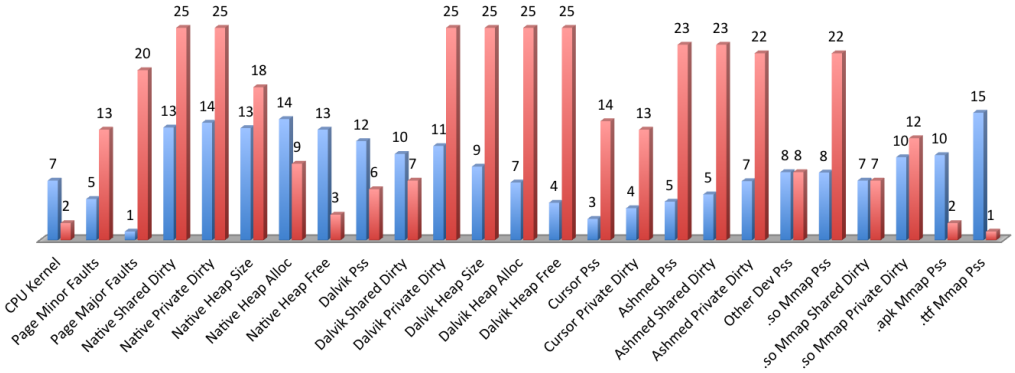

Figure 5.4. Average ranking of the features that appear among the top 15 the most indicative ones. Blue color depicts the average rank of the observed features, while red color depicts its number of occurrences.

shared by different processes and the page is already brought into memory for other processes. *Dalvik Heap Size* is the RAM used by Dalvik allocations in the applications.

5.2 Malware Detection

Since malware detection is one of the main goals of the proposed methodology, in this section we outline the results we obtained in this domain, the most indicative features for malware detection task, obtained detection accuracy in detection of malicious records, malicious groups of records and malicious application. Also, since one of the main design principles of the method that we propose is to take into account power consumption and detection time in addition to the detection accuracy, in this section we also discuss the results obtained while designing such a detection system, dependencies between the three requirements taken into account and the obtained optimized solutions.

5.2.1 Identification of the most indicative features for malware detection

After preliminary analysis of indicative features characterized by PCA, described in Section 5.1, further investigation of memory features importance for malware detection task was performed using other feature selection techniques discussed in Section 4.4 and the dataset discussed in Section 4.1.3. The first 15 highest ranked features for the used feature selection methods are shown in Table 5.1.

We may notice that different ranking methods provide different results; however, a number of features related to memory mapping, such as *.ttf mmap PSS* and *.so mmap PSS*, are among the highest ranked ones in most methods. In Table 5.1 for CFS Subset Evaluator only seven features are listed, without any ranking. This is due to the fact that CFS does not evaluate single features but, instead, subsets of features, calculating their usefulness with respect to the other subsets.

Table 5.1. Top 15 features ranked by different feature selection algorithms.

Correlation Attribute Evaluator		**CFS Subset Evaluator**	**Gain Ratio Attribute Evaluator**	
Ranking	*Feature*	*Feature*	*Ranking*	*Feature*
0.49	.ttf mmap PSS	.so mmap Shared Dirty	0.12	.jar mmap PSS
0.46	.so mmap PSS	.jar mmap PSS	0.06	.dex mmap Private Dirty
0.43	.dex mmap PSS	.ttf mmap PSS	0.05	Other dev Private Dirty
0.39	TOTAL PSS	.dex mmap PSS	0.04	.dex mmap PSS
0.38	.so mmap Shared Dirty	.dex mmap Private Dirty	0.04	.ttf mmap PSS
0.36	TOTAL Private Dirty	Other mmap Private Dirty	0.04	.so mmap Shared Dirty
0.36	.jar mmap PSS	CPU Total	0.03	.so mmap PSS
0.33	Unknown Private Dirty		0.03	.so mmap Private Dirty
0.33	Unknown PSS		0.03	Other mmap Private Dirty
0.31	CPU User		0.03	.dex mmap Shared Dirty
0.30	CPU Total		0.03	Unknown PSS
0.30	TOTAL Heap Size		0.03	Native Heap Size
0.29	TOTAL Heap Alloc		0.03	Unknown Private Dirty
0.27	Other mmap PSS		0.03	Other mmap Shared Dirty
0.26	Native Heap Alloc		0.02	TOTAL PSS

Information Gain Attribute Evaluator		**OneR Attribute Evaluator**	
Ranking	*Feature*	*Ranking*	*Feature*
0.28	.dex mmap PSS	79.99	.ttf mmap PSS
0.27	.ttf mmap PSS	79.64	.dex mmap PSS
0.25	Unknown PSS	78.92	Unknown PSS
0.25	.so mmap PSS	78.51	.so mmap PSS
0.23	Native Heap Size	76.24	TOTAL Heap Size
0.23	TOTAL Heap Size	76.19	Native Heap Size
0.21	Unknown Private Dirty	75.87	Unknown Private Dirty
0.21	.so mmap Private Dirty	75.44	.so mmap Private Dirty
0.17	.so mmap Shared Dirty	73.25	.so mmap Shared Dirty
0.17	Ashmed PSS	71.26	Native PSS
0.15	Other mmap PSS	71.21	Other mmap PSS
0.15	TOTAL PSS	71.17	Ashmed PSS
0.14	minor faults	70.75	minor faults
0.13	Native PSS	69.78	TOTAL PSS
0.13	TOTAL Private Dirty	69.49	.apk mmap PSS

It is our observation that the behavior of memory mapping is very important in the discrimination between malicious and benign program executions. In fact, as it can be also seen in Table 5.1, all of the five applied feature selection methods list the importance of mmap related features very high in the ranking. In particular, in case of CFS Subset Evaluator that in our scenario performs the best,

out of seven features selected as the most indicative, six are related to memory and more precisely to memory mapping aspects. Related to this, we can further observe that both shared and private memory aspects play significant role in the discrimination of malware and benign software, so as Dalvik executable files, true type fonts and Java archive related aspects. Finally, what the dynamic parameter that also influences the detection results is the CPU total consumption of the applications under analysis.

5.2.2 Runtime Detection of Malicious Records

As we can observe from Table 5.1 different feature selection methods identify different features as the most indicative. In order to validate the effectiveness of the identified indicative features in detection of malicious execution records and maximize the F-Measure, we have used the three detection algorithms, introduced in 3.3.1 having different approach to detection: NB, LR, and J48 Decision Tree.

The validation of the classification is done by using ten-fold cross validation, that is widely used model validation technique that can estimate how accurately observed model will perform in practice, as pointed out by Kohavi [1995]. In case of ten-fold cross validation, the dataset is divided into ten parts, where at each round, one part, consisting of nine folds, is considered as a training set and the remaining part is used as a test set. This procedure is repeated ten times, each time using a different training and a test set, and after ten rounds the average detection performance is reported (precision, recall and F-measure).

In case of NB and LR we have evaluated the results of the classifiers both considering all the available features and by considering only the highest ranked ones for each feature selection method. We have repeated this procedure for all five feature selection methods. More in detail, CFS Subset Evaluator is applied by using Greedy forward search through the space of attribute subsets. Correlation Attribute Evaluator, Gain Ratio Attribute Evaluator, Information Gain Attribute Evaluator, and OneR Attribute Evaluator are used together with Ranker method that ranks attributes by their individual performance. For J48 Decision Tree, we have trained the model by changing the maximum number of instances that a node can have and by observing its quality of detection (F-measure). The number of instances is important in order to avoid over-fitting, a situation in which the decision tree consists of many nodes that are very well fitted for the training set, but do not perform well on a test set and unseen data. In J48 Decision Tree, the maximum number of instances in the tree can be set, but indicative features are selected by the algorithm itself and cannot be set externally. NB

has been set assuming normality and modelling each conditional distribution with a single Gaussian; LR has been used with ridge estimator, since it has been shown by Le Cessie and Van Houwelingen [1992] that it can improve attribute estimation and decrease the error made in further predictions. J48 has been used with pruning, setting the confidence factor used for pruning to 0.25 and the minimum number of instances per leaf to 5000; these values represent a trade-off between number of instances in the dataset, speed of execution and the quality of classification.

Table 5.2 summarizes the results obtained with these three classifiers. The initial model includes all the 53 features available. The second model, instead, is the one providing maximum F-measure; the last one provides the best ratio between F-measure and the number of features considered.

Table 5.2. Performance of the classifiers when different number of features is considered.

		Model		
		Initial	**Max. F-measure**	**Optimized**
NB	Precision	0.79	0.84	0.84
	Recall	0.76	0.83	0.83
	F-measure	0.77	0.83	0.83
	No. of features	53	7	7
LR	Precision	0.84	0.86	0.84
	Recall	0.84	0.86	0.84
	F-measure	0.83	0.86	0.84
	No. of features	53	38	7
J48 Decision Tree	Precision	–	–	0.83
	Recall	–		0.83
	F-measure	–	–	0.82
	No. of features	–	–	6

In case of NB, with the dataset containing all 53 features, an F-measure of 0.77 is obtained. After taking into account rankings of the features for all five feature selection methods, we identified an optimized set of features, both with respect to their number and to F-measure, as the one computed with the CFS Subset Evaluator. As it can be seen from Table 5.2, the obtained F-measure in this case is 0.83, with a set of seven features.

In case of LR, using all features, we obtained F-measure of 0.83. However, decreasing the number of features and observing system performance, we obtained an increased F-measure of 0.86, obtained by using Correlation Attributes with rankings higher than 0.03 (38 features). The optimal model with respect to both number of features and quality of detection is again obtained by using features selected by CFS Subset Evaluator. This model has an F-measure of 0.84.

For what concerns J48, the obtained model, that uses six features, has an F-measure of 0.82. The decision tree, with the selected features thereby used, is shown in Figure 5.5.

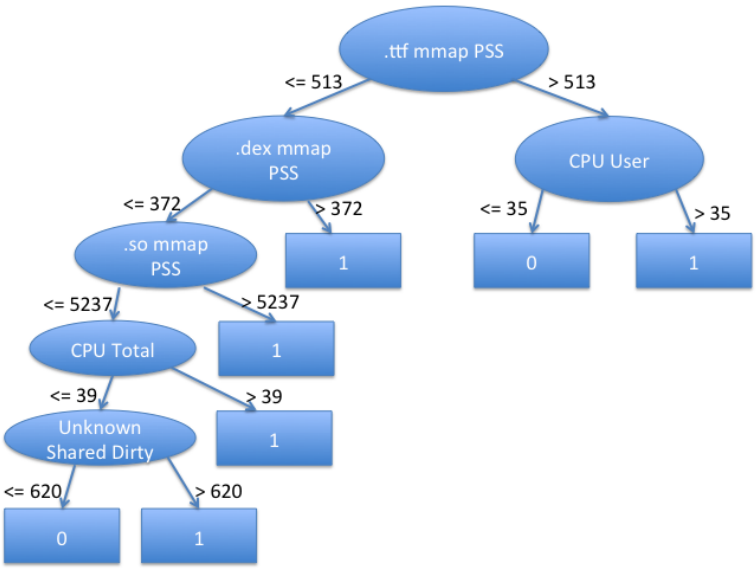

Figure 5.5. Optimized J48 Decision Tree; malicious and benign traces are labelled with 0 and 1, respectively.

In summary, features that are the most indicative for LR and NB classifiers are the seven listed in the *CFS Subset Evaluator* column of Table 5.1; for J48 Decision Tree, the six most indicative features are listed in Figure 5.5. Out of seven the most indicative features six are related to the memory consumption usage of the observed application and different aspects of memory mapping and one to their CPU consumption. These features are described in Table 5.3.

In summary, the best detection performance with respect to F-measure, so as with respect to the ratio between F-measure and the number of features considered, is achieved by the LR algorithm. The best F-measure is obtained by considering 38 highest ranked features provided by the Correlation Attribute Evaluator; the optimized set of only 7 features is obtained by considering the CFS Subset Evaluator. Also the other two detection algorithms considered, namely NB and J48 Decision Tree, perform well, showing similarly good detection performance.

We have shown that in case of LR and NB, detection performance, measured with precision, recall, and F-measure, and presented in Table 5.2 is improved, even if the number of features is decreased from 53 to 7. Reducing the number

Table 5.3. Brief description of the most indicative features Google Inc. [2015].

Category	Feature Name	Feature Description
Memory mapped native code	.so mmap Shared Dirty	Shared memory, in the *Dirty* state, being used for mapped native code. The *Dirty* state is due to fix-ups to the native code when it is loaded into its final address
Memory-mapped Dalvik code	.dex mmap PSS	Memory usage for Dalvik code, including pages shared among processes
	.dex mmap Private Dirty	Private memory, in the *Dirty* state, being used for Dalvik code. The *Dirty* state is due to fix-ups to the native code when it is loaded into its final address
Memory-mapped fonts	.ttf mmap PSS	Memory usage for true type fonts, including pages shared among processes
CPU usage	CPU Total	Total (User + System) CPU usage by the considered application
Other memory-mapped files and devices	Other mmap Private Dirty	Private memory used by unclassified contents that is in the *dirty* state
	.jar mmap PSS	Memory usage for Java archives, including pages shared among processes

of features without reducing performance of the system is in line with our intention to design effective, while at the same time efficient, mobile detection system. We have observed that malicious samples consume less memory and CPU compared to the benign ones. This can be seen looking into values of features of J48 Decision Tree presented in Figure 5.5. The reason for this could be that malicious applications perform only a limited activity, connected to their malicious intent, rather than the legitimate actions for which they were installed by the user.

From the results of our experiments, we can conclude that CPU and memory features contain useful information for discriminating between the execution traces related to malicious or benign applications. Additionally, good detection performance can be obtained while using algorithms of low complexity, suitable for battery-operated mobile devices.

5.2.3 Detection of Malicious Sub-traces

In order to detect malicious sub-traces, we use the approach proposed in Section 3.3.3, and take into account dynamic features related to memory, CPU, network behavior, as well as corresponding system calls.

We focus on traces in which the period of observations is one observation every 2 s: the number of system calls generated during the corresponding time frame is in the order of some hundreds. The clustering step is performed using the KMeans++ algorithm discussed by Arthur and Vassilvitskii [2007] that partitions the observations into k clusters such that each observation belongs to the cluster with the nearest mean that is used as a prototype of the cluster. For

each cluster, a classifier is built using the method proposed by Canfora, Medvet, Mercaldo and Visaggio [2015], based on sequences of system calls and RF classification, due to its reported high detection performance tested on a significant portion of existing malware.

The considered features related to memory and CPU information are explained in 4.3. The set of system calls considered in this work is based on the one successfully used by Canfora, Medvet, Mercaldo and Visaggio [2015] and it is composed of following system calls: `msgget`, `getpid`, `ioctl`, `recv`, `semget`, `getuid32`, `mprotect`, `SYS_224`, `read`, `syscall_983042`, `write`, `gettimeofday`, `writev`, `sigprocmask`, `mmap2`, `munmap`, `close`, `lseek`, `brk`, `pread`, `fstat64`, `open`, `dup`, `fcntl64`, `stat64`, `getdents64`, `access`, `clone`, `semop`, `getpriority`, `fsync`, `nanosleep`, `_llseek`, `unlink`, `lstat64`, `pwrite`, `chmod`, `rename`, `sched_yield`, `pivot_root`, `mkdir`, `ipc_subcall`, `getsockopt`, `getcwd`, `pipe`, `sched_getsched`, `sched_getparam`, `socket`, `uname`, `getgid32`, `getegid32`, `geteuid32`, `ftruncate`, `syscall_317`, `select`, `rmdir`, `connect`, `bind`, `flock`, `setsockopt`, `getsockname`, `kill`, `fork`.

The experiments we have performed are taking into account different distance metrics for clustering stage, different number of clusters and different number of trees for classification stage with RF. We have experimented with number of clusters 3, 5, and 7 and number of trees ranging from small (5 trees) to high (50 trees). Concerning the time interval T which defines the lasting of a sub-trace under analysis, we used the interval of $T = 40\,\mathrm{s}$, which corresponds to 20 observations for each sub-trace, since it represents in our opinion a good trade-off between time in which malware can expose its malicious intents (if the interval is too long the malicious behavior could spread) and false positive rate (if the interval is too small could create too many false positives).

We took into account following distance metrics for KMeans++ clustering:

- *Euclidean distance* is the most commonly used distance metric that represents straight-line distance between two points in an Euclidean space; the distance between two points $\vec{x}$ and $\vec{y}$ is given by $d(\vec{x},\vec{y}) = \sqrt{\sum_i (x_i - y_i)^2}$.
- *Canberra distance* has already been successfully used for intrusion detection, as discussed by Emran and Ye [2002]; it is given by $d(\vec{x},\vec{y}) = \sum_i \frac{|x_i - y_i|}{|x_i| + |y_i|}$.
- *Chebyshev distance* between two vectors is the largest of their differences along any coordinate dimension; it is given by $d(\vec{x},\vec{y}) = \max_i |x_i - y_i|$.

We have used 1,709 benign and 1,523 malicious applications for training and testing, respectively. The evaluation has been performed by using 3 fold cross

validation; in each round, sub-traces belonging to 1,000 benign and 1,000 malicious applications have been used as a training set; the remaining ones are used as a test set. This implies that testing is always performed on applications that are previously unseen by the detection system. Detection accuracy is obtained by averaging the results obtained in the three rounds.

The obtained results in term of sub-trace classification accuracy are shown in Table 5.4.

Table 5.4. Sub-trace accuracy.

			Observation		
Distance	No of clusters	No of trees	FPR	FNR	Acc.
Euclidean	3	5	0.12	0.61	0.63
Euclidean	3	50	0.33	0.44	0.61
Euclidean	5	5	0.14	0.53	0.66
Euclidean	5	50	0.17	0.51	0.66
Euclidean	7	5	0.14	0.58	0.64
Euclidean	7	50	0.28	0.39	0.67
Canberra	3	5	0.20	0.54	0.63
Canberra	3	50	0.60	0.23	0.58
Canberra	5	5	0.24	0.46	0.65
Canberra	5	50	0.19	0.49	0.66
Canberra	7	5	0.42	0.28	0.65
Canberra	7	50	0.54	0.16	0.65
Chebyshev	3	5	0.07	0.78	0.58
Chebyshev	3	50	0.07	0.70	0.62
Chebyshev	5	5	0.43	0.34	0.62
Chebyshev	5	50	0.49	0.29	0.61
Chebyshev	7	5	0.54	0.28	0.59
Chebyshev	7	50	0.31	0.35	0.67

From these results we can draw a set of conclusions. First, increasing the number of trees in RF classifier, does not necessary increase detection performance. This is promising because it means that the method can be applied without need for algorithms of high complexity. Additional conclusion is that increasing the number of clusters does increase detection performance. In our opinion this is the case because the higher number of clusters is able to preserve better the diverse behavior of malware. Furthermore, there is no dominantly better distance, although the Euclidean one provides higher accuracy also when

smaller numbers of clusters are considered. The highest accuracy of detection being 0.67 is achieved with both Euclidean and Chebyshev distance with 7 clusters and 50 trees. While this number is not great in absolute, we find it useful to perform an initial marking of the potentially malicious sub-traces, that can then be further investigated in more detail.

5.2.4 Malicious applications detection

As introduced in Section 3.3.2, the classification algorithm of choice has to be compatible with the limited resources of mobile devices. Based on the results we obtained in the malicious records detection from the Table 5.2 and based on their complexity, we have adopted LR as our execution records classification algorithm. However, since our detection system is modular, any other technique or method, with satisfying detection performance and low complexity, could be used in this case.

In order to validate the approach proposed in Section 3.3.2, we divided the collected execution traces of the dataset into a training, a testing, and a validation set, consisting of 1,298 (727 malware, 571 benign), 579 (304 malware, 275 benign), and 183 samples (89 malware, 94 benign), respectively. The samples in the testing and validation set contain malware samples belonging to the same malware families as in the training set, but the samples themselves are different (previously unseen by the detection system in the training procedure). Motivation for such an approach is that most of existing new malware samples on the market are actually repackaged ones coming from previously existing malware families, as also state by Zhou and Jiang [2012].

The algorithm parameters that we use for malicious applications classification have been chosen experimentally by considering the test dataset. Parameters taken into account are: the sliding window length, the threshold on the number of records classified as malware in the window, and the number of disjointed sliding windows that need to be marked as malware.

Due to the fact that these parameters have impact on final classification results, different values have been explored in two phases: a coarse-grain exploration, used to determine the most promising values for the parameters, and a fine-grain exploration, used to explore the neighborhood of the most promising values determined in the first phase. The values considered in the two exploration phases are shown in Table 5.5. The exact values depicted in Table 5.5 are related to the length of execution of the observed malicious and benign applications, that in our experiments was 10 minutes, as previously described in Section 4.2. The main motivation behind the introduction of such parameters in

the development of detection system was to investigate what is the best way to capture the malicious behavior at runtime, if it is better to look at long patterns that appear rarely (i.e., longer window length and lower number of checks), short patterns that appear frequently (i.e., shorter window length and higher number of checks), or something in between that reflects the best the intentions of the running programs. Formally optimizing the values of the window length, threshold and the number of checks is an open problem and requires a separate treatment. In this work we took an experimental approach. In the observed scenario, larger values for the sliding window length parameter are possible, but, the larger the window, the longer the time needed to detect malware, especially considering that multiple windows should be identified as malicious for classifying an application as malware. Designing a detection system in this way allows us to have a direct impact on detection accuracy and detection time of the system, and to be able to customize it for a specific application scenarios if needed. Execution records are available, as previously mentioned, with a periodicity of two seconds.

The validation of the classification is first performed by using ten-fold cross validation, and then, by separating the dataset into training, testing, and validation sets.

Table 5.5. Algorithm parameters used in the two phases of the exploration.

Parameter	**Coarse-grain exploration**	**Fine-grain exploration**
Sliding window length	5, 10	1, 3, 5, 10, 15, 20
Threshold (%)	60, 70, 80, 90	60, 65, 70, 75, 80, 85, 90, 95
Checks (no.)	1, 2, 5, 7, 10	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

The analysis was done by considering both the malicious records classification obtained with the *Initial* set of features (all considered 53 features) and the one obtained with the *Optimized* set (features selected as the most indicative for malicious records detection that are described in Table 5.3). Three different metrics have been used for choosing the best set of parameters:

- highest F-measure
- best malware detection rate, with false positive rate below 30%
- lowest false positive rate with malware detection rate greater than 70%.

In the coarse-grain exploration phase, all the combinations of the parameters shown in the leftmost part of Table 5.5 have been considered. In Table 5.6 the

values that provide best results in this exploration phase are shown; the optimal values differ for the different sets of features as well as for the different metrics considered.

Table 5.6. Best malicious applications detection results obtained in the coarse-grain exploration of detection algorithm's parameters.

Feature set	Metric	Window length, Threshold, Checks	Malware detected %	False positives %	F-measure
Initial	Highest F-measure	5, 60, 7	95.7	31.6	0.85
	Best malware detection	5, 60, 10	92.1	27.3	0.85
	Lowest false positive	10, 70, 10	71.4	17.7	0.76
Optimized	Highest F-measure	10, 90, 7	90.1	18.2	0.87
	Best malware detection	10, 60, 7	95.7	29.1	0.86
	Lowest false positive	10, 90, 10	72.4	9.8	0.80

Based on the results of the coarse-grain exploration, we defined new ranges of values for the fine-grain exploration; the values considered in this phase are listed in rightmost part of Table 5.5. The results obtained by using these parameters are shown in Table 5.7. In some cases, similar results, in terms of F-measure and true/false positive, have been obtained with different sets of parameters with very similar values. In these cases, we have chosen the set of parameters with the smallest window length, due to the associated shorter detection time. Results obtained from the fine-grain exploration show slight improvements in the considered metrics. With the optimized model, the detection rate and the false positive rate are equal to 92.1% and 19.3%, respectively. On the average, malware is detected after 81 execution records that, with the 2s sampling period for monitoring infrastructure that we used in our experiments, corresponds to 2 minutes and 42 seconds from the beginning of the execution of each sample.

As far as algorithm parameters are concerned (sliding window length, threshold, and number of checks), we can observe that a higher detection rate is obtained by using medium-size windows with a low threshold and a relatively low number of checks. Conversely, to minimize false positives, larger windows with high thresholds should be preferred. The best F-measure is achieved by considering small windows with a not-too-high threshold, but with a high number of checks.

Validation of the detection system has been performed by using execution traces generated from previously unseen applications (i.e., malicious and benign applications that have been used neither during training nor testing). Algorithm parameters considered during validation are the ones that have been identified as the best during the fine-grain exploration phase.

Table 5.7. Best malicious applications detection results obtained in the fine-grain exploration of detection algorithm's parameters.

Feature set	Metric	Window length, Threshold, Checks	Malware detected %	False positives %	F-measure
Initial	Highest F-measure	3, 80, 11	96.0	31.3	0.86
	Best malware detection	5, 60, 9	93.7	28.7	0.85
	Lowest false positive	15, 95, 5	72.0	14.5	0.78
Optimized	Highest F-measure	5, 85, 15	92.1	19.3	0.88
	Best malware detection	10, 60, 7	95.7	29.1	0.86
	Lowest false positive	20, 90, 5	71.4	9.4	0.79

Results of validation are shown in Table 5.8. As we can see in this scenario, on the previously unseen samples the obtained higest F-measure is of 0.85, where 85.5% of malware is correctly identified as such with 17.2% false positive rate. Having in mind the low number of features in the *Optimized* set (seven), so as the fact that the samples considered have never been seen by the classifier during training and testing, we can observe that validation shows satisfactory detection performance.

Table 5.8. Best malicious applications detection results attained on the validation dataset with the parameters obtained from fine-grain exploration.

Feature set	Metric	Window length, Threshold, Checks	Malware detected %	False positives %	F-measure
Initial	Highest F-measure	3, 80, 11	92.1	24.5	0.84
	Best malware detection	5, 60, 9	89.9	23.4	0.84
	Lowest false positive	15, 95, 5	65.2	10.6	0.74
Optimized	Highest F-measure	5, 85, 15	85.5	17.2	0.85
	Best malware detection	10, 60, 7	89.9	24.7	0.83
	Lowest false positive	20, 90, 5	59.5	10.7	0.70

Detection speed is strictly related to the parameters chosen for malicious applications detection: larger sliding windows and higher number of checks introduce a longer delay in detecting malware. When the *Optimized* features set is considered with the parameters that maximize F-measure, malicious applications are identified, on the average, after about 85 execution records; which makes the detection time of 2 minutes and 50 seconds.

The discussed results have been obtained by using previously unseen malware samples. We believe that even better results, in terms of detection accuracy, could be obtained if, along with execution traces of previously unseen applications,

alternative execution traces of known applications were considered during test and validation. While we decided not to do it, to be consistent with statistical analysis consolidated practices, we believe that this condition would be closer to the one found in reality, where just a small number of malware samples are completely unknown to the detection mechanisms.

5.2.5 Optimization of Sampling Period of Monitoring Infrastructure and Power Consumption

In order to find a suitable trade-off between the detection accuracy, detection time and power consumption, as introduced in Section 3.3.4, in addition to testing our malicious applications detector with different sliding window algorithm parameters, listed in Table 5.5, we also performed the analysis of its detection performance using different sampling periods for the monitoring infrastructure: 2s, 4s, 6s, 8s, 12s, and 16s.

In order to understand the relation between the most suitable parameters of our sliding window based detection mechanism and considered sampling periods, we first investigate how F-measure changes if we change sampling period without changing (re-tuning) the parameters of the sliding window. This relation between F-measure and the sampling period depicted in Figure 5.6, shows us that in order to keep F-measure high the selection of the sampling period has to be accompanied with the selection of the suitable sliding window algorithm parameters. This can be seen in cases of all considered sampling periods, but is particularly noticeable in case of 2s sampling period (red line in Figure 5.6) where once sampling period is increased, detection accuracy drops significantly. In general, when unsuitable parameters are used for a certain sampling period, detection performance, expressed by F-measure, degrades rapidly. At the same time, if the parameters are correctly chosen, detection performance is not significantly affected by the increased sampling period.

Having in mind the important connection between the sampling period and the selection of the most suitable parameters for sliding window mechanism in order to achieve high F-measure, we have investigated what are the most suitable parameters for each of the observed sampling periods, by repeating the procedure for the malicious applications detection described in Sections 3.3.2 and 5.2.4. The results of this analysis are shown in Table 5.9 where best detection performance obtained with the test set when different sampling periods are considered. If we look at the changes of the most suitable sliding window parameters and the sampling period, we can observe that with the increased sampling pe-

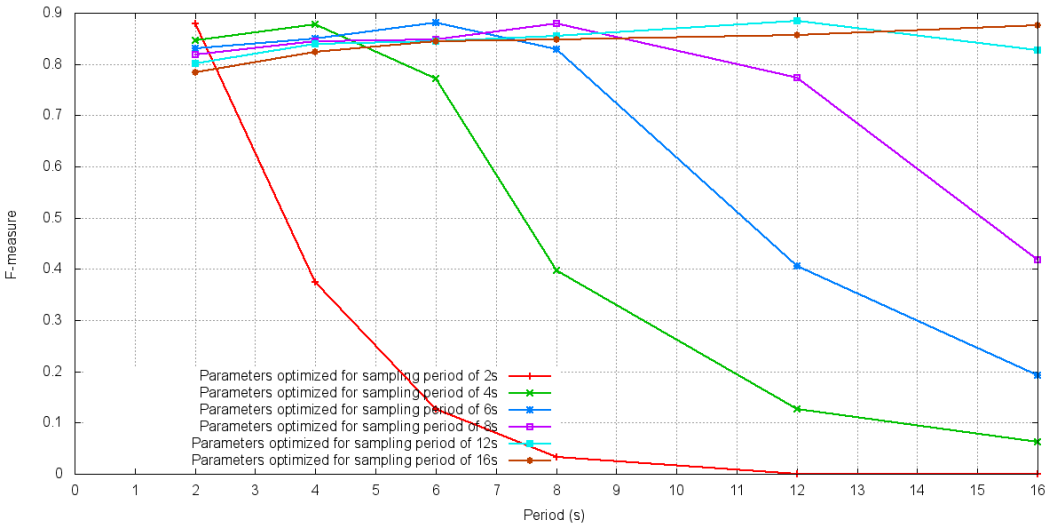

Figure 5.6. F-measure obtained with different sampling periods and the sliding window algorithm parameters that maximize F-measure.

riod, the parameters of the sliding window mechanism that the best detect presence of malware tend to decrease. In addition to the obtained F-measure and detection time, in this evaluation we also include the measured battery power consumption, that was collected in the way introduced in Section 4.5. Power consumption is a parameter that is directly connected with the sampling period, and that decreases when the sampling period increases, as it also can be observed from the results obtained in Table 5.9. Our analysis showed that the parameters of the sliding window mechanism, different window sizes, thresholds and different number of checks, do not affect the consumed power in the current setup of the experiments and that it changes significantly only in the relation to the sampling period.

Table 5.9. Best results obtained in the algorithm parameters exploration when different sampling periods are considered.

Sampling period (s)	Window size	Threshold	No. of checks	F-measure	Detection time (s)	Average Power (mW)
2	5	80	15	0.88	161.94	54
4	3	70	12	0.88	150.36	32
6	3	70	8	0.88	149.52	24
8	3	70	6	0.88	147.36	20
12	3	70	4	0.88	143.52	16
16	3	70	3	0.87	137.76	14

In addition to the evaluation of the proposed malicious applications detection on the test set, we evaluate its performance also on the validation set with the unseen applications and in Table 5.10 we show the obtained results. The setting of the sliding window parameters are the same as the ones enclosed in Table 5.9. From the obtained results we can see that when suitable detection parameters are chosen, the F-measure variation is much less than the one depicted in Figure 5.6. Even-more, the F-measure in this situation varies only slightly, from 0.85 up to 0.83. Detection time, on the unseen applications, varies from 159.84s up to 172.12s.

Table 5.10. Validation with different sampling periods.

Sampling period (s)	Window size	Threshold	No. of checks	F-measure	Detection time (s)	Average Power (mW)
2	5	80	15	0.85	172.12	54
4	3	70	12	0.84	162.68	32
6	3	70	8	0.84	161.64	24
8	3	70	6	0.84	165.2	20
12	3	70	4	0.83	159.84	16
16	3	70	3	0.83	163.2	14

Further insights on the dependencies between F-measure, detection time and power can be seen in Figure 5.7, where we show the results in terms of F-measure, power, and detection time obtained when all the possible combinations of window length, number of checks, thresholds, and sampling periods are considered. While in this graph we show all the possible combinations of parameters, even the ones that we have defined as unacceptable with too low detection rate (i.e., below 70%) and/or too high false positive rate (i.e. greater than 30%) in Figure 5.8, instead, we outline only those configurations that produce acceptable results. By acceptable results we consider only the ones that have detection rate above 70% and false positive rate below 30%, since lower values than this would not be practical for any real-time scenario. As it can be seen, in this case the number of acceptable solutions decreases when the sampling period is increased.

The dependencies among the three parameters – detection accuracy, detection time and power consumption – when the sliding window parameters and the sampling period are changed, can be better seen in Figures 5.9, 5.10, and 5.11 where they are plotted in pairs, so that it is easier to understand their relation.

Figure 5.9 shows F-measure and power of all the configurations with acceptable performance. It can be noticed that all power levels, associated with sampling periods, have a configuration that reaches similar detection performance, even though the lower the sampling period the higher the number of valid con-

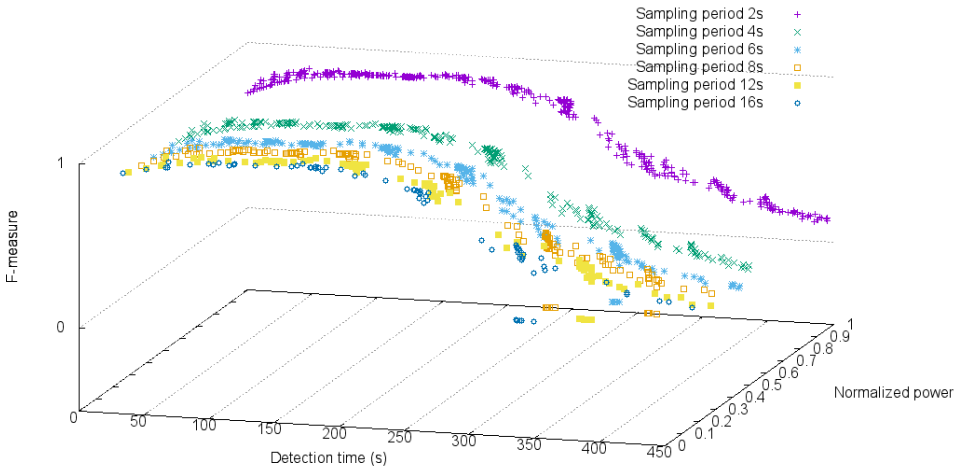

Figure 5.7. F-measure, power, and detection time obtained for all configurations (sliding window algorithm parameters and sampling periods).

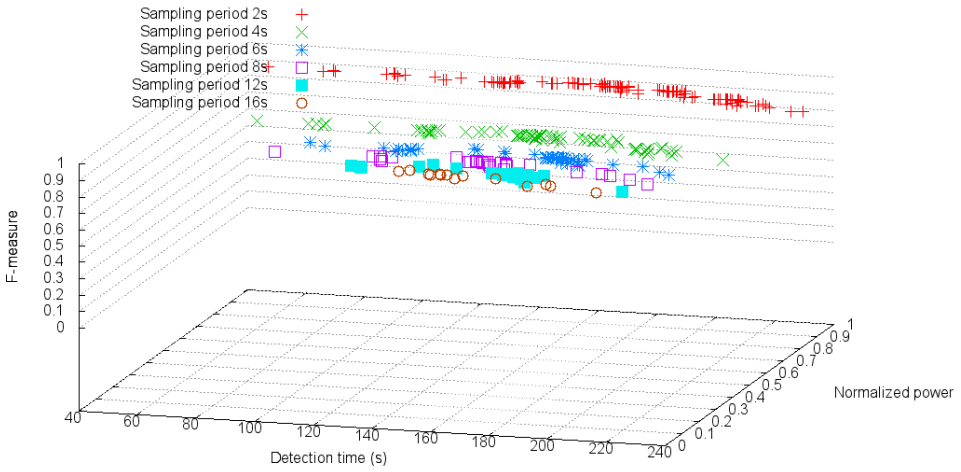

Figure 5.8. F-measure, power, and detection time obtained for configurations (sliding window algorithm parameters and sampling periods) that have acceptable performance.

figurations to chose from. Since bigger sampling periods, associated with lower power levels, can provide similar detection performance as the ones provided by smaller sampling periods, from this point of view higher sampling periods should

be favoured.

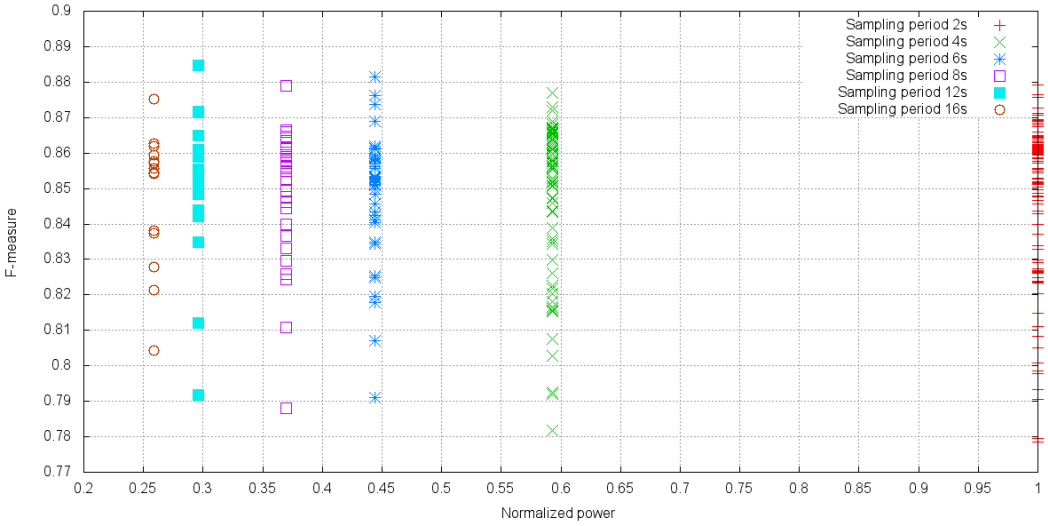

Figure 5.9. F-measure vs. power for all the configurations that have acceptable performance.

Figure 5.10 shows F-measure and detection time of all the configurations with acceptable performance; it can be noticed that F-measure reaches its maximum with detection times between 140 and 160 seconds. Detection times longer than 190 seconds are associated with solutions with the worse F-measure. From the point of view of the detection time, in this case there is no generally preferred selection, since the valid solutions are spread among all observed sampling periods.

Figure 5.11 shows the relationship between power and detection time of all the configurations with acceptable performance. As previously noted, lower power configurations are associated with higher sampling periods but have less available configurations. We can notice how detection times belonging to the configurations with the same power consumption tend to concentrate in groups related to the parameters used. We can also conclude that if we prefer shorter detection time, the lower sampling periods should be chosen. In fact, if the shortest detection time has the priority in the applications domain, for the scenario that we observed, the decision on the possible infection could be obtained after 47s.

As previously shown in Section 3.3.5, comparing different solutions can be difficult when optimizing for multiple parameters, for which we introduced the metric shown in Equation 3.1.

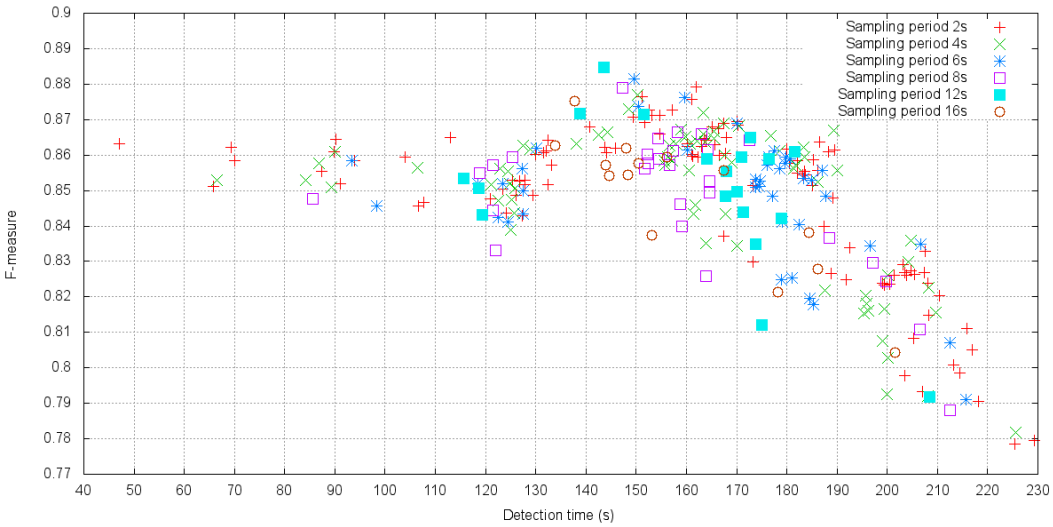

Figure 5.10. F-measure vs. detection time for all the configurations that have acceptable performance.

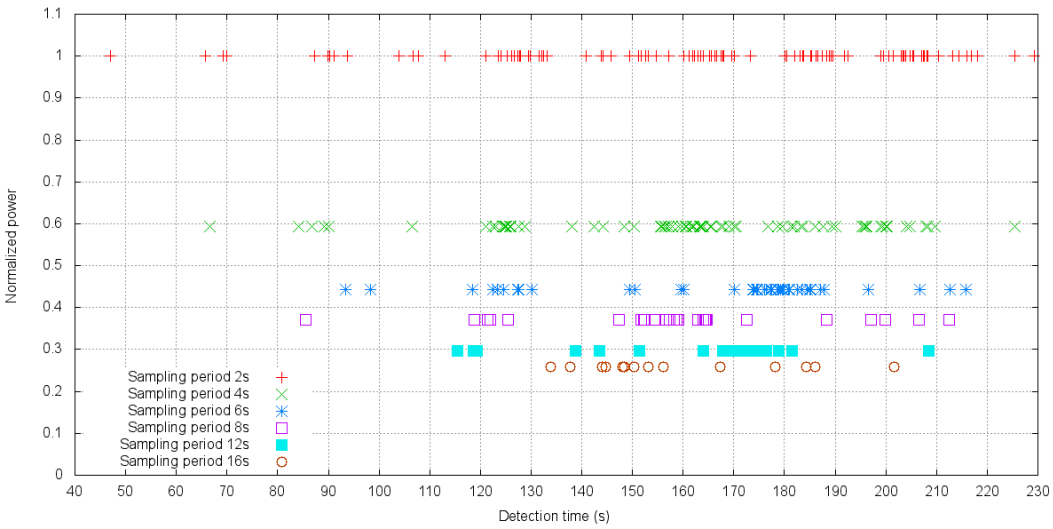

Figure 5.11. Power versus time for all the configurations that have acceptable performance.

Figure 5.12 shows the values of the metric for all the valid configurations. It can be noticed how values of the metric tend to increase by increasing the sampling period from 2s to 16s: this is due to the decrease in power, since, as previously discussed, detection time and F-measure are similar for solutions using different sampling periods of the monitoring infrastructure. The configu-

rations that maximize our metric are the ones with sampling periods 8, 12, and 16 s. In particular, the solutions that maximized the metric has a sampling period of 8 s, with window size equal to 10, threshold equal to 90% and number of checks equal to 1. With this solution, F-measure is equal to 0.85, detection time is equal to 85.52 s and power is equal to 20mW; false positive and detection rates are equal to 28% and 92%, respectively. The selected detection solution is a result of the optimization between detection accuracy, detection time and power consumption, when each of these requirements is taken into account as equally important. In some application-specific scenarios such detection performance might not be the most suitable. For example, the detection time of 85.52 s for some types of malware might be too long, or the detection rate of 92% could not be tolerated. By having designed malware detection system that takes into account different requirements, we can further tune it by considering those application-specific requirements that are the most important in the observed detection environment, such as best detection time, best F-measure, lowest false positive, or highest detection rate. Although we do not investigate in detail such extreme application-specific scenarios, in Table 5.11 we list the configurations that satisfy such criteria and that could be used in scenarios when the mentioned requirements have the highest significance. Even though we use the metric that takes mentioned requirements as equally important, also other metrics that favour one parameter over the others (e.g., by using the square values of the parameter itself) can be adopted with different results in terms of the selected optimized configuration, but with the steps of the methodology performed in a same way.

Table 5.11. Best configurations for the observed optimization criteria.

Optimized for	Sampling period (s)	Window size / Threshold / No. of checks	F-measure	False positive / Detection rate	Detection time (s)	Average Power (mW)
F-measure	12	3 / 70 / 4	0.88	21.45% / 94.73%	143.52	16
Detection time	2	20 / 95 / 1	0.86	28.36% / 95.39%	47.06	54
Lowest fp (detection > 30%)	4	50 / 95 / 1	0.79	8% / 70.39%	200	31
Highest detection (fp<30%)	12	1 / 60 / 12	0.87	26.9% / 96.05%	138.84	16

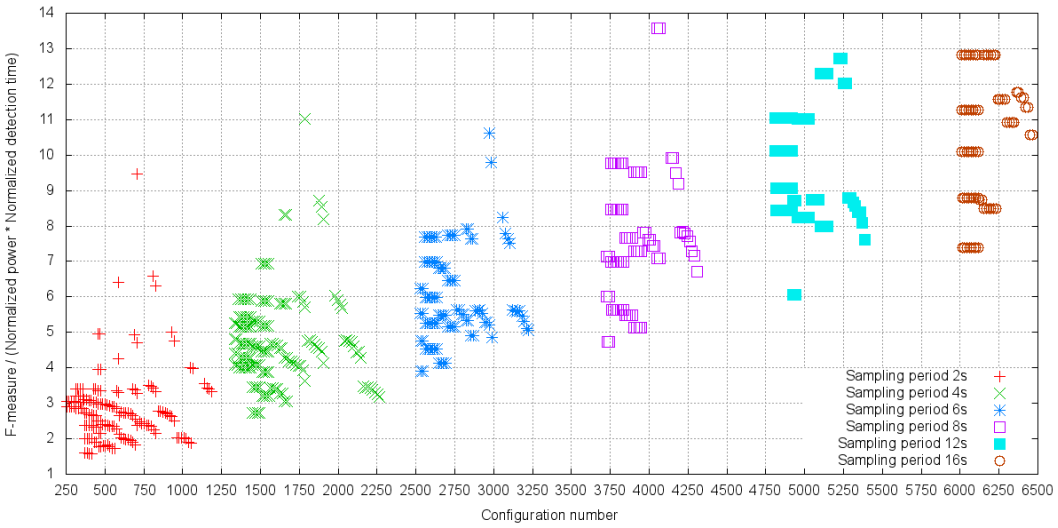

Figure 5.12. Metric values of all the configurations that have acceptable performance.

5.3 Malware Classification

In this section we describe the results we obtained when the approach on discrimination between diverse malware behavior introduced in Section 3.4 is applied to features representing memory, CPU and network behavior. In order to evaluate the effectiveness of the proposed approach, the Android malware families described in Section 4.1.4 are used. We selected all considered malware families to be Trojans, because of the fact that the detection of Trojans is a particularly difficult task, due to their similarity with legitimate, benign, applications. At the same time, Trojans are one of the most present malware types in mobile devices, so their detection is of high importance to the community and relevant for the users.

The information about number and types of the applications used in the evaluation of the method are described in Section 4.1.4. Both malware and benign samples have been executed in the Android SDK in a way introduced in Section 4.2. As a classification algorithm, apart from the previously described NB and LR classifiers, we also used SVM, that is a supervised learning model that given a set of training examples, each marked for belonging to one of two categories, builds a model that assigns new examples into one category or the other. The SVM implementation suggested by Platt [1999] and available in Weka was used; this implementation includes sequential minimal optimization. The selected kernel is the commonly used polynomial one, that allows learning of non-linear models.

In the remaining of this section we discuss the selection of the indicative features for different malware families, performed by CFS Subset Evaluator, and the results obtained using these features and classifiers of low complexity in discrimination between them, that show how memory, network and CPU contain enough information to perform malware classification.

In this scenario all the collected features related to the memory, CPU, and network behavior are used. In total, this makes 73 features. The goal of the performed experiments was to observe if the difference in behavior between benign and malicious trojanized execution records is reflected by their different influence on memory, CPU and network behavior and if there are differences between Trojans belonging to different malware families. In order to investigate if there are behavioural differences among different malware families, in the performed experiments we marked as *malware* only the samples belonging to the considered Trojan family and all the others as *benign*, and we evaluate the detection performance of the used algorithms with respect to that. As in malware detection scenario, we used ten fold cross-validation for validating the obtained results.

Out of the selected sets of the most indicative features per family, we create a superset that is formed by the union of the sets of features identified as the most significant ones for each family. This superset is the one that is monitored and stored on device and sent for remote analysis in case of malware detected. The features in this superset are listed and described in Table 5.12, where they are also described. The obtained results using these features and the classifiers of interest are enclosed in Table 5.13.

The most informative features for the Droid Kung Fu family, as shown in Table 5.12, are three; by considering these three features and the LR classifier, according to Table 5.13, we are able to achieve a detection accuracy equal to 76.5% on detection of malicious records. Also in the case of the Fake Player family, three features are selected as the most indicative ones and malware in this family can be detected with the highest accuracy of 99.7% using LR, while, at the same time, both NB and SVM show similarly good detection performance. Also the Geinimi family can be detected accurately by using SVM or LR. The most indicative features that have been selected are 13 (5 related to memory usage, 3 related to CPU consumption, and 8 related to network traffic). Considering only these features we have achieved an accuracy equal to 97% by using SVM classifier. Six features have been selected for the Ginger Master Trojan family. By using these features, the highest obtained detection accuracy, using SVM, is 78.7%. For the Kmin Trojan family, eight memory-related, one CPU-related, and two network-related features have been selected as the most significant ones,

Table 5.12. Representative features of the observed malicious Trojan families

		Trojan Family				
Feature Name	**Description**	Droid Kung Fu	Fake Player	Geinimi	Ginger Master	Kmin
.ttf mmap PSS	Memory usage for true type fonts, including pages shared with other processes	×			×	×
Network load over last minute	Network load in bits/second registered in the last minute	×	×		×	
Maximum packet size in bytes	Maximum network packets size in bytes observed in the 2s sampling period	×				
.dex mmap Pss	Memory usage for Dalvik code, including pages shared with other processes		×			
Number of ARP packets	Number of ARP network packets received and transmitted in the 2s sampling period		×			
.jar mmap Pss	Memory usage for Java code, including pages shared among processes			×		
Other mmap Pss	Memory usage for non-classified purposes, including pages shared among processes			×		
Other mmap Shared Dirty	Memory shared among processes, in the *Dirty* state, being used for non-classified purposes			×		
Number of TCP packets	Number of TCP network packets received and transmitted in the 2s sampling period			×		
Unknown Pss	Memory usage for unknown purposes, including pages shared among processes			×		
TOTAL Heap Size	Total heap size allocated for the process			×		×
CPU User	User-space CPU usage of the process			×		
CPU kernel	Kernel-space CPU usage of the process			×		
minor faults	Virtual memory minor page faults caused by the process			×	×	×
bps	Network load in bits/second registered in the 2s sampling period			×	×	×
Number of ICMP packets	Number of ICMP network packets received and transmitted in the 2s sampling period			×		
Size in byte standard deviation	Standard deviation of the network packet size in bytes in the 2s sampling period			×		
Number of bytes	Number of bytes transmitted and received in the 2s sampling period			×		
.so mmap Private Dirty	Private memory of the process, in the *Dirty* state, being used for mapped native code				×	×
Number of UDP packets	Number of UDP network packets received and transmitted in the 2s sampling period				×	×
Ashmem Private Dirty	Private memory of the process, in the *Dirty* state, being allocated as Android shared memory					×
.so mmap Pss	Memory usage for mapped native code, including pages shared with other processes					×
.so mmap Shared Dirty	Memory shared with other processes, in the *Dirty* state, being used for mapped native code					×
.apk mmap Pss	Memory usage for Android application package files, including pages shared with other processes					×
TOTAL Shared Dirty	Total memory of the process that is shared and it is marked as dirty					×

Table 5.13. Obtained malware classification results

	Droid Kung Fu	**Fake Player**	**Geinimi**	**GinMaster**	**Kmin**
NB	58%	99.7%	90%	77.2%	81.5%
LR	76.5%	99.8%	96.8%	78.4%	95.3%
SVM	74.2%	99.7%	97%	78.7%	95.4%

and using them the highest detection accuracy of 95.4% is achieved by again using SVM classifier.

From the performed experiments and by comparing the selected features per each malware family, we can observe that different behaviors of different malicious Trojan families are reflected by different usage patterns of memory, CPU, and network. The behavior of some families (Kmin) is better reflected in memory related features; the behavior of some other families (Droid Kung Fu, Fake Player) is, instead, better reflected through network-related features; for some other families (Geinimi), features from all categories are important for detection. The differences are significant enough to allow us to identify these families with good accuracy. We validated this observation by using three different classifiers and showed that high detection accuracy can be achieved using only a reduced set of features. From Table 5.12 we can see that some malware families have common representative features. This means that if we use a dedicated classifier for identification of each family, it could happen that more than one of them identifies running malware as belonging to that particular family. In real-time usage such problem would be resolved by taking as an output of the classification the result from a classifiers for which, based on the offline analysis, we know that the confidence in detection of the observed malware family is the highest.

5.4 Performance Evaluation of a Hybrid Method

In this section, we report the results obtained when considering static detection alone, dynamic detection alone, and the hybrid method that combines these two as described in Section 3.5. For the evaluation of this approach we have used a dataset consisting of malicious applications and benign applications described in Section 4.1.5.

5.4.1 Static Detection Results

We first discuss the obtained results when only static detection method described in Section 3.5.1 is used. As mentioned in Section 3.5.1 we have experimented with three different classifiers, namely J48, Naive Bayes (NB), and Logistic Regression (LR) as well as with n equal to 2. We have used $h = 50$ as parameter for feature selection. After the learning phase, performed by using the training set described in Section 4.1.5, we applied the obtained classifier, C, to each application of the test set and we measured precision, recall, and F-Measure.

Obtained results are shown in Table 5.14. With the three different classification algorithms considered in the study, we have obtained a precision ranging from 0.968 to 0.998, a recall ranging between 0.988 and 0.997, and the F-Measure between 0.980 and 0.998. The classifier with best performance was J48.

Table 5.14. Classification results for malicious and benign applications when features extracted by static analysis are considered along with the J48, NB and LR classifiers.

Algorithm	**Precision**		**Recall**		**F-Measure**	
	Malware	Benign	Malware	Benign	Malware	Benign
J48	0.998	1.000	0.997	1.000	0.998	1.000
NB	0.968	1.000	0.992	0.998	0.980	0.999
LR	0.994	0.999	0.988	1.000	0.991	0.999

The static method classifies correctly all the benign applications (i.e., no benign application is marked as malware), but nine malicious applications are misclassified as benign. In other words, the static method has no false positive, but it has nine false negatives. False negatives are represented by three malware samples from the *Simple Locker* family and six samples from *Koler* family; all the samples from the other families (i.e., *Locker*, *Svpeng*, and *ScarePackage*) are classified correctly as malware.

5.4.2 Dynamic Detection

As described in Section 3.5.2, we applied our dynamic detection method in this scenario, where we considered dynamic features related to memory, CPU usage and features derived from network usage and statistics on system calls. In Table 5.15 we enclose the classification results obtained by using the algorithms for the detection of malicious records, the first stage of our malicious applications detection system. While NB provides lower detection accuracy, both LR and J48

perform well as shown by all considered metrics. Although both of these methods are of low complexity and suitable for on-device detection, due to its ability to assign a probability of being malicious to a record, instead of just providing a binary decision, we opted for the Logistic Regression classifier as a detector of malicious records also in this scenario.

Table 5.15. Classification results for malware and benign applications when features extracted by dynamic analysis are considered along with the J48, NB and LR classifiers.

Algorithm	**Precision**		**Recall**		**F-Measure**	
	Malware	Benign	Malware	Benign	Malware	Benign
J48	0.988	0.994	0.976	0.997	0.982	0.995
NB	0.382	0.975	0.940	0.605	0.543	0.747
LR	0.933	0.973	0.894	0.983	0.913	0.978

After the classification of the malicious records is done, we use the sliding window technique to investigate if complete applications are malicious or benign. In order to find the most suitable parameters for this scenario, we again run a batch of experiments by considering the initial combinations of parameters depicted in the leftmost part of the Table 5.5 on the training set. Based on these parameters, we have selected the configurations that provide highest F-measure, highest detection accuracy with false positive below 20%, lowest false positive with accuracy higher than 80%, and lowest detection time. The best configurations according to the aforementioned metrics are shown, along with the corresponding detection performance, in Table 5.16. In these results, detection time is measured from the first execution record marked as malicious. We have performed a fine-grain exploration in the surroundings of the optimal parameters identified during the coarse-grain exploration. The combinations of parameters considered in this phase are shown in rightmost side of the Table 5.5. The results of this exploration, which lead to the optimal parameters of our dynamic detection system, are shown in Table 5.17; as previously mentioned, the parameters can be chosen to optimize different metrics and this is reflected in the table. The best trade-off between these requirements is represented by highest F-measure that in our case is 0.85. This configuration provides high accuracy, with a low number of false positive and a short detection time.

Based on these obtained results, we have selected the configurations that provide highest F-measure, highest detection accuracy with false positive below 20%, lowest false positive with accuracy higher than 80%, and lowest detection time. These configurations are considered having in mind different possible application scenarios (those in which detection accuracy would have the highest

priority, those where the lowest false positives would have the highest priority, and those in which balance of both of them would be preferred).

Table 5.16. Best results obtained in coarse-grain exploration.

Configuration	Window length	Threshold (%)	Checks	Detection rate (%)	False positive rate (%)	F-measure	Detection time (s)
Highest F-measure	10	70	1	91.28	3.78	0.84	21.24
Highest detection rate / lowest detection time	5	60	1	93.57	5.33	0.81	9.28
Lowest false positive	10	70	1	91.28	3.78	0.84	21.24

Table 5.17. Best results obtained with respect to the observed metrics.

Configuration	Window length	Threshold (%)	Checks	Detection rate (%)	False positive rate (%)	F-measure	Detection time (s)
Highest F-measure	9	78	1	90.82	3.46	0.85	20.46
Highest detection rate / lowest detection time	1	60	1	96.33	19.84	0.58	0
Lowest false positive	10	70	2	80.27	2.92	0.80	60.84

After selecting the optimal parameters on the training set, we tested them on the test set. In Table 5.18 we show the obtained results. As expected, detection performance decreases with respect to the training set, but it still remains high; for example, when the parameters for highest F-measure are selected, the detection rate decreases from 90% to 85%. The case of the parameters chosen for the lowest false positive rate provides best performance on the test set, with an increase of the detection rate and a slight increase of the false positive rate. We can observe that results outlined in Table 5.8 in the scenario in which we were using the balanced dataset introduced in Section 4.1.3 are in line with the results obtained in this, hybrid scenario, where we are using an unbalanced dataset introduced in Section 4.1.5.

Table 5.18. Results obtained in the testing phase with the best parameters obtained in the exploration phase. Detection time is measured from the first record identified as malicious.

Configuration	Window length	Threshold (%)	Checks	Detection rate (%)	False positive rate (%)	F-measure	Detection time (s)
Highest F-measure	9	78	1	85.61	5.31	0.88	24.24
Highest detection rate / lowest detection time	1	60	1	93.03	25.06	0.79	0
Lowest false positive	10	70	2	84.68	3.81	0.89	44.72

When best F-measure parameters are considered, undetected malware in the test set is equal to 62. By relying on the Virus Total [n.d.] analyses, we have classified these samples in categories; we have taken as a reference F-secure since it provides descriptive malware definitions. Most of the malicious samples that are not identified by our dynamic malware detection method belong to the Simplelocker (61%) and to the Koler (19%) families. The remaining samples were not identified as malicious by F-Secure. Similarly to static detection and although providing promising results on unseen malicious samples, our dynamic detection alone was not able to detect all the observed malicious samples.

5.4.3 Hybrid Detection

To evaluate the effectiveness of the hybrid approach, we have considered the list of malicious applications in the test set that are not detected by the used static method, and we have checked whether our dynamic detector could correctly detect them as malware. In fact, all the nine applications that are not detected by the used static method are correctly identified as malware by our dynamic method, with all the three sets of optimal parameters. Therefore, we have verified our initial assumption that by using a hybrid method we could increase the coverage of malware detection. In fact, starting from a detection rate of 99.8% for static detection and of 85.61% (with best F-measure parameters) for dynamic detection, we obtain a 100% detection rate for hybrid detection. While the static method has no false positives, the dynamic method has some, as shown in Table 5.18. Considering the extremely good detection performance, we can choose to optimize the dynamic method for the lowest number of false positives, which when the corresponding parameters from Table 5.18 are chosen, makes the false positive rate of 3.81%. In summary, the results show that using the hybrid method is the way to follow in order to provide effective protection against malware not only for its increased coverage, but also to unite the accuracy of static detection with the possibility of detecting malware at runtime.

5.5 Limitations of the Proposed Approach

Our detection system uses dynamic features that are by their nature more difficult to evade than the static ones. The assumption for usage of dynamic features is that even when the applications are repackaged or obfuscated, during their execution they will still have similar behavioral footprints, and a classifier trained on these features could properly discriminate malicious applications from benign

ones. Our detection method can detect effectively known malicious samples and unknown malicious samples belonging to known malware families. However, it cannot guarantee, as any other detection method, absolute security. If an attacker develops a new malicious sample that has a behavior reflected in significantly different feature profiles than the ones observed in our dataset, the detection system might not be able to detect it.

Our approach is data driven and, as other methods that use a similar approach, such as the one proposed by Shabtai et al. [2012] and by Ham and Choi [2013], it does not provide any help in understanding how the selected features and their relations are connected with the behavior of the applications. Therefore, while selected features can be described individually and the rationale beyond algorithms can be explained, an explanation on why features are important for the identification of malware is possible only in the simplest cases, where the number of available features is very limited and the system is extremely simple. However, without usage of these algorithms and models it would be impossible to measure usefulness of features, identify their correlations, and choose only the most indicative ones.

Additionally, our approach, being based on dynamic detection, inherits the same weakness of other dynamic detection methods when it comes to malicious payload triggering. Namely, in our experiments we were executing malicious applications with Monkey runner tool, and, in order to trigger and record their malicious intents and to stimulate the malicious payload we used a set of activation events proposed by Zhou and Jiang [2012]. However, we cannot guarantee that the malicious execution path is actually triggered within the observed execution time. This is a common problem of dynamic detection methods where many possible execution paths exist and it is challenging to guarantee that exactly the one with malicious intents is triggered.

In comparison to static analysis, our method shows lower detection accuracy, and the hybrid method outperforms both individual approaches. However, we need to point out that static and dynamic detection are different in what they offer. In fact, static detection is able to detect malware before the application containing it is executed, thereby preventing the malicious payload from executing and, thus, preventing any harm for the system. Dynamic detection, instead, detects malware while applications are being executed and, thus, when some harm to the system might already been caused. As previously discussed, our detection method is relatively fast in detecting malicious behavior, but still it leaves open the possibility of harming the system. On the other hand, dynamic detection is able to capture malware at runtime, thus offering protection against dynamically installed code, which cannot be detected by static methods since it

is run with much lower periodicity, and also protection against malicious actions hidden in the obfuscated or encrypted code.

5.6 Comparison of the Obtained Results with State-of-the-art Methods

The main differences between our work and the other approaches on mobile malware detection on devices, are:

- In Section 3.1 and in Milosevic, Malek and Ferrante [2017], we propose a methodology for malware detection and malware classification, aimed to be used on devices with limited computational capabilities.

- In Section 3.3 and in Milosevic et al. [2014] we presented a malware detection architecture aimed to be used at runtime on mobile devices, that not only considers detection accuracy, but also detection time, and power consumption as main requirements.

- We created and used a dataset consisting of dynamic features related to memory, CPU, network behavior and system calls, that, to the best of our knowledge, was not considered up to date.

- In Milosevic, Malek and Ferrante [2016] and in Section 5.2.2, by means of feature selection methods, we reduced the set of features from 53 to 7, without affecting detection performance. Although the detection accuracy is slightly lower than in the approaches presented by Canfora, Medvet, Mercaldo and Visaggio [2015], by Ham and Choi [2013], and by Dini et al. [2012], the number of features considered in our approach is much smaller, which results in lower computational overhead and reduces the size of the monitoring infrastructure

- Our malicious applications detection method presented in Milosevic, Ferrante and Malek [2016a] uses linear Logistic Regression and a simple sliding window technique with a lower complexity than the approaches proposed by Ham and Choi [2013], by Dini et al. [2012], and by Canfora, Medvet, Mercaldo and Visaggio [2015]

- Our detection system outperforms, in terms of detection accuracy, four representative mobile anti-virus software on Malware Genome dataset. The

detection results of the mentioned anti-virus software that are obtained by Zhou and Jiang [2012] are enclosed in Section 2.2.1

- In Milosevic, Ferrante and Malek [2016b] and in Section 5.3 we identified features to be monitored for classification of different malicious Trojan families, based on which, using ensemble of detection algorithms of low complexity, we could discriminate execution records belonging to different families. While the approaches presented in state-of-the are effective in identifying malware, to the best of our knowledge, no dynamic approach exists that is able to perform such discrimination with such a small number of features

- As opposed to currently existing methods, that focus on the identification of complete mobile applications as malicious or benign, the method we proposed in Ferrante et al. [2016] and in Section 3.3.3 can be used also to detect malicious parts of applications executions

- In Milosevic, J. and Ferrante, A. and Malek, M. [2017] and in Section 5.2.5 we investigate the correlations between detection time and detection accuracy, detection accuracy and power consumption as well as between power consumption and detection time, when the system parameters of our malicious applications detection system are changed, and we investigate the importance of changing the sampling period with respect to these parameters. To the best of our knowledge, such comprehensive evaluation of dependencies between detection time, detection accuracy and power consumption is not performed up to date

- Most of the existing malware detection methods is either focused on only static or only dynamic analysis, and only few approaches consider hybrid detection. In Ferrante et al. [2017] and in Section 5.4 we evaluate the detection performance of a hybrid method that combines static and dynamic approach, that is, to the best of our knowledge, the first work with the aim to automatically identify real-world Android malware by using a hybrid approach in order to assure a very high detection ratio.

Also, several results and their extensions have been published in several papers such as: Milosevic et al. [2014]; Milosevic, Ferrante and Regazzoni [2015]; Milosevic, Ferrante and Malek [2016c]; Milosevic, Malek and Ferrante [2016]; Milosevic, Ferrante and Malek [2016a,b]; Ferrante et al. [2016]; Milosevic, Malek and Ferrante [2017]; Ferrante et al. [2017] and in our internal report Milosevic, J. and Ferrante, A. and Malek, M. [2017].

Chapter 6

Conclusions and Future Work

Widespread adoption of smart, mobile devices helps their users in many different domains and brings to them enormous possibilities previously unseen in human era. Some of many domains in which users already profit from these devices are smart homes, smart cities, smart grids, health-care and assistive technologies. However, while more and more of the critical tasks are offloaded to machines and intelligent systems, we have to make sure that their smartness is accompanied with their secure design and that they are, in addition to providing envisioned service, also able to protect users against unwanted malicious threats or at least effectively detect them when/if they happen. However, due to the constantly increasing complexity of the connected systems and a variety of possible attack scenarios, this is a challenging task, that is further complicated by the fact that the devices are battery-operated and with limited computational resources.

In order to further facilitate the adoption of smart mobile devices and enable their secure usage, we focused on design, development and validation of the effective and efficient runtime malware detection system. We aimed at detection method suitable for the resource-constrained environment and able to early detect potential malicious infections, and due to this, we proposed a detection system that takes into account in addition to detection accuracy, also detection time and power consumption. By using these three figures of merit we evaluated a set of different detection system parameters, and we identified those that provide the optimized solution with respect to the observed metric. Our results show that, if we design detection systems while from the early beginning having in mind not only detection accuracy, but also detection time and power consumption, we can provide effective detection solution that is at the same time efficient enough to fit the limited resources of mobile devices. In the rest of this chapter we first summarize the obtained results and then we outline the envisioned

future steps.

6.1 Conclusions

The main goal of our work was to provide effective malware detection solutions suitable to be used at runtime on resource-constrained devices. In order to achieve this goal we observe the influence of malicious and benign applications on dynamic system parameters and based on this influence we learn our detection systems to discriminate malware from benign execution records, malicious from benign groups of execution records and malicious from benign applications. The detection systems we trained also learned to discriminate between different types of malware based on the way malicious applications belonging to diverse malware families interact with the observed dynamic parameters, namely based on their behavioural patterns. Finally, our developed system is able to do this while at the same time taking into account the resource constrained environment of mobile devices, particularly constrained power, and also while having early detection time as one of its priorities.

Our results show that, by observing only a limited number of features related to memory and CPU (seven in case of NB and Logistic Regression, and six in case of J48 Decision Tree), the execution records belonging to malicious applications can be identified with precision and recall of more than 85%. Additionally, on the average, a previously unseen malware application can be detected within the first 2 minutes and 45 seconds of execution and the discrimination between previously unseen malicious and benign applications can be performed with an F-measure of 0.85 and with sampling period of two seconds. This can be achieved with a sliding window technique that we proposed and that has linear complexity, and that is suitable for on device usage. According to our results, one of the most important aspects to be considered in such a design, that is currently missing in the state of the art, is the selection of the most suitable sampling period. If sampling period of the monitoring infrastructure is increased and with respect to it we retrain our detection systems, we can still obtain a satisfying detection performance, while decreasing the consumed power. Namely, the sampling period that shows the best performance for the considered scenario is of eight seconds, with which the F-measure of 0.85 is obtained, detection time is 85.52s, power is 20mW, and with false positive of 28% and detection rate of 92%. The accuracy of detection of malicious sub-traces with our proposed method is 0.67. This method takes into account, apart from memory and CPU information, also system calls that can be collected at runtime. Using behavioral patterns we proposed,

we demonstrated that we can discriminate between different types of execution records belonging to diverse malware families with accuracy of up to 99.8%. These behavioral patterns capture discriminative influence of different malware families on dynamic features related to memory, CPU and network behavior, and can be identified with high accuracy using a small set of representative features. The aforementioned results show the detection performance of our approach in a scenario where all applications are being analyzed only dynamically by using the approach we propose. However, when we evaluated our detection method in a hybrid scenario, where the applications are first pre-screened for the presence of malware offline using static analysis, and then analyzed at runtime by our method, the detection accuracy that we obtained was of 100% and the false positives rate was 3.81%, showing that the combination of two approaches (static and dynamic) is the best way to protect the devices against malware.

6.2 Future Work

Increasing number of battery-operated mobile devices and their increasing applications in all aspects of humans life, rise the need for their secure operation and importance for their protection against malicious threats. Due to this, we envision that in the future the relevance of the malware detection for resource-constrained devices will continue to increase and we intend to continue working in this area where effective solutions are still needed in order to provide help and protection against malware to billions of users and to empower them to take the advantage of smart connected devices in the best possible way. Following, we describe the main research questions that we aim to tackle:

6.2.1 Are static or dynamic features more resistant against unseen malware families?

When it comes to the detection of malware, dynamic detection is more resistant against unknown samples since it looks at their behavior at runtime that is not as easy to obfuscate as their source code, which is usually what is analyzed by static methods. However, if the malware is coming from a completely new malware family, previously unseen in the training stage, it is not clear what would be the detection performance of dynamic methods and which one would perform better, static or dynamic method. One of our envisioned future works is to investigate this. In order to achieve this goal, we aim at performing the analysis in which training would be done using a set of families released up to certain

time and testing would be done with a set of families released after the considered time (i.e., training with families released up to year 2015 and testing with families released after), and then on such a dataset we would evaluate the detection performance of both static and dynamic analysis and compare the obtained results.

6.2.2 Is deep learning a good candidate for malware classification task, and if yes, can we have timely decisions using it?

In our currently proposed architecture, the features indicative for the presence of the investigated malware families are stored at runtime, and, when the presence of malware is indicated, they are sent into a network for the further analysis of the type of malware that is being detected on the phone. In this domain, we identified a set of the most representative features for the observed families, and verified their importance in the discrimination of malware by performing the analysis described in Section 5.3. However, this detection is still at the level of the execution records. The next challenge that we want to address is the extension of this module to the application level. In the envisioned scenario, the classification will take place at the cloud infrastructure, and for each of the families that we observe in the analysis, we envision to include comprehensive set of countermeasures to be taken, if its identified with high confidence. Since the analysis in this case would happen in a non resource-constrained environment, we aim at use of different sophisticated algorithms to further increase the accuracy of detection, with higher complexity than currently used ones, that are used for the on device detection. As one of the potential candidates for such analysis we foresee a classifier based on deep learning, since deep learning have outperformed, in terms of detection accuracy, many other state-of-the-art machine learning based methods and since it was already successfully used in different domains where high detection accuracy matters. However, in addition to the high complexity of deep learning based methods, that makes them suitable only for the off-device analysis, the main question and the main challenge is to evaluate if the inference time is short enough to make their classification outcome useful and applicable in the practical scenario. The investigation of this research problem and the answer to this question is one of our envisioned future steps.

6.2.3 Would we benefit from a dedicated hardware accelerators for on-device malware detection?

Sending information into cloud infrastructure for further analysis is one way to follow in order to avoid running complex, power and memory consuming, algorithms on a device, that are already constrained in terms of these resources. The additional way to cope with this problem is through design and development of dedicated hardware accelerators, that would be used to perform the most computationally complex part of the calculations on the device. However, while different software solutions are proposed in the state of the art, the same trend is not followed in the hardware design domain. Bridging this gap and designing a hardware accelerator for the further increase in the efficiency of malware detection is also one of the our envisioned future steps.

6.2.4 What are the performance of our detection methods if applied in the other elements of the IoT infrastructure?

The methodology we propose takes into account the constrained environment of mobile devices by minimizing the number of monitored features, by adopting algorithms of low complexity, and by using the sampling period that is the trade-off between detection performance, detection time and consumed power. We evaluated the performance of the proposed methodology by using malware samples for Android OS and Nexus 5, a mobile device with reasonable computing power. However, we believe that this methodology can be successfully applied also on other IoT devices, that are even more constrained in terms of available resources and it is one of our envisioned future steps to investigate this hypothesis.

6.2.5 Using this methodology can we detect cyber attacks in smart grid?

While electrical grid becomes smart, it is at the same time becoming exposed to different cyber attacks and thus more vulnerable. On the other side, since smart grid is a relatively new concept, there are not that many solutions for the effective protection against attacks in them. One of the envisioned future steps is the application of the proposed methodology in the domain of the smart grid and the evaluation of the detection performance in that environment.

In the future *everything smart and connected world* there will be many problems, certainly some related to security. However, if we design our systems with security in mind and we adopt detection time and power consumption from early design as equally important factors as the detection accuracy, then we can already now mitigate some of those possible problems and deliver *smart, secure and connected systems*.

Bibliography

2016 Threats Prediction [2015]. *Technical report*, McAfee Labs.

Aafer, Y., Du, W. and Yin, H. [2013]. DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android, *Security and Privacy in Communication Networks - 9th International ICST Conference, SecureComm 2013, Sydney, NSW, Australia, September 25-28, 2013, Revised Selected Papers*, pp. 86–103.
URL: *http://dx.doi.org/10.1007/978-3-319-04283-1*$_6$

Android Open Source project [2015a]. Android Debug Bridge. Online: http://developer.android.com/tools/help/adb.html.

Android Open Source project [2015b]. Android Software Development Kit. Online: https://developer.android.com/sdk/index.html.

Android Open Source project [2015c]. UI/Application Exerciser Monkey. Online: http://developer.android.com/tools/help/monkey.html.

Andronio, N., Zanero, S. and Maggi, F. [2015]. Heldroid: Dissecting and detecting mobile ransomware, *International Workshop on Recent Advances in Intrusion Detection*, Springer, pp. 382–404.

Antunes, N. and Vieira, M. [2015]. On the metrics for benchmarking vulnerability detection tools, *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015, Rio de Janeiro, Brazil, June 22-25, 2015*, pp. 505–516.
URL: *https://doi.org/10.1109/DSN.2015.30*

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H. and Rieck, K. [2014]. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket, *NDSS*.

Arp, D., Spreitzenbarth, M., Huebner, M., Gascon, H. and Rieck, K. [2014]. Drebin: Efficient and explainable detection of android malware in your pocket,

Proceedings of 21th Annual Network and Distributed System Security Symposium (NDSS).

Arthur, D. and Vassilvitskii, S. [2007]. k-means++: The advantages of careful seeding, *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, pp. 1027–1035.

Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D. and McDaniel, P. [2014]. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps, *SIGPLAN Not.* **49**(6): 259–269.
URL: *http://doi.acm.org/10.1145/2666356.2594299*

Becher, M., Freiling, F. C., Hoffmann, J., Holz, T., Uellenbeck, S. and Wolf, C. [2011]. Mobile security catching up? revealing the nuts and bolts of the security of mobile devices, *Symposium on Security and Privacy*, SP '11, IEEE Computer Society, pp. 96–111.

Bläsing, T., Batyuk, L., Schmidt, A.-D., Camtepe, S. A. and Albayrak, S. [2010]. An android application sandbox system for suspicious software detection, *Malicious and unwanted software (MALWARE), 2010 5th international conference on*, IEEE, pp. 55–62.

Bose, A., Hu, X., Shin, K. G. and Park, T. [2008]. Behavioral detection of malware on mobile handsets, *6th international conference on Mobile systems, applications, and services (MobiSys)*, ACM, pp. 225–238.

Breiman, L. [2001]. Random forests, *Mach. Learn.* **45**(1): 5–32.

Burguera, I., Zurutuza, U. and Nadjm-Tehrani, S. [2011]. Crowdroid: Behavior-based malware detection system for android, *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, ACM, New York, NY, USA, pp. 15–26.

Canfora, G., De Lorenzo, A., Medvet, E., Mercaldo, F. and Visaggio, C. A. [2015]. Effectiveness of opcode ngrams for detection of multi family android malware, *Availability, Reliability and Security (ARES), 2015 10th International Conference on*, IEEE, pp. 333–340.

Canfora, G., Medvet, E., Mercaldo, F. and Visaggio, C. A. [2015]. Detecting android malware using sequences of system calls, *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, ACM, pp. 13–20.

Canfora, G., Mercaldo, F. and Visaggio, C. A. [2013]. A classifier of malicious android applications, *Proceedings of the 2nd International Workshop on Security of Mobile Applications, in conjunction with the International Conference on Availability, Reliability and Security*.

Canfora, G., Mercaldo, F. and Visaggio, C. A. [2015]. Evaluating op-code frequency histograms in malware and third-party mobile applications, *E-Business and Telecommunications*, Springer, pp. 201–222.

Cheng, J., Wong, S. H., Yang, H. and Lu, S. [2007]. Smartsiren: virus detection and alert for smartphones, *5th international conference on Mobile systems, applications and services*, MobiSys '07, ACM, pp. 258–271.

Cisco 2014 Annual Security Report [2014]. *Technical report*, Cisco.
URL: *https://www.cisco.com/web/offer/gist*$_t y 2_a sset/Cisco_2 014_A SR.pdf$

Clark, S. and Fu, K. [2012]. Recent results in computer security for medical devices, *in* K. Nikita, J. Lin, D. Fotiadis and M.-T. Arredondo Waldmeyer (eds), *Wireless Mobile Communication and Healthcare*, Vol. 83 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer Berlin Heidelberg, pp. 111–118.

Contagio Dataset [2016]. Online: http://contagiodump.blogspot.ch.

Delac, G., Silic, M. and Krolo, J. [2011]. Emerging security threats for mobile platforms, *MIPRO, 34th International Convention*, IEEE, pp. 1468–1473.

Dimjaševic, M., Atzeni, S., Rakamaric, Z. and Ugrina, I. [2016]. Evaluation of android malware detection based on system calls, *ACM Conference on Data and Application Security and Privacy (CODASPY)*.

Dini, G., Martinelli, F., Saracino, A. and Sgandurra, D. [2012]. Madam: A multi-level anomaly detector for android malware, *MMM-ACNS*, Vol. 12, Springer, pp. 240–253.

Dunham, K. [2008]. *Mobile Malware Attacks and Defense*, Elsevier Science, Syngress.

Emran, S. M. and Ye, N. [2002]. Robustness of chi-square and canberra distance metrics for computer intrusion detection, *Quality and Reliability Engineering International* **18**(1): 19–28.
URL: *http://dx.doi.org/10.1002/qre.441*

Enck, W., Ongtang, M. and McDaniel, P. [2009]. On lightweight mobile phone application certification, *16th ACM conference on Computer and communications security (CCS)*, ACM, pp. 235–245.

Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M. and Rajarajan, M. [2015]. Android security: a survey of issues, malware penetration, and defenses, *Communications Surveys & Tutorials, IEEE* **17**(2): 998–1022.

Felt, A. P., Finifter, M., Chin, E., Hanna, S. and Wagner, D. [2011]. A survey of mobile malware in the wild, *1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM)*, ACM, pp. 3–14.

Felt, A. P., Greenwood, K. and Wagner, D. [2011]. The effectiveness of application permissions, *2nd USENIX conference on Web application development (WebApps)*, USENIX Association.

Ferrante, A., Malek, M., Martinelli, F., Mercaldo, F. and Milosevic, J. [2017]. Extinguishing Ransomware - a Hybrid Approach to Android Ransomware Detection, *The 10th International Symposium on Foundations Practice of Security*.

Ferrante, A., Medvet, E., Mercaldo, F., Milosevic, J. and Visaggio, C. A. [2016]. Spotting the malicious moment: Characterizing malware behavior using dynamic features, *The Fifth International Workshop on Security of Mobile Applications (IWSMA) 2016*, CPS, Salzburg, Austria.

Gartner [2017]. Press release. Online: http://www.gartner.com/newsroom/id/3598917.

Google Inc. [2015]. Android Developers – Investigating Your RAM Usage. Online: `http://developer.android.com/tools/debugging/debugging-memory.html`.

Google Play Store [2015].
URL: *Online: https://play.google.com/store?hl=en*

Group, C. [2015]. 2015 Cyberthreat Defense Report, *Technical report*. Online: `http://www.brightcloud.com/pdf/cyberedge-2015-cdr-report.pdf`.

Guo, C., Wang, H. J. and Zhu, W. [2007]. Smart-phone attacks and defenses.

Hall, M. A. [1998]. *Correlation-based Feature Subset Selection for Machine Learning*, PhD thesis, University of Waikato, Hamilton, New Zealand.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I. H. [2009]. The WEKA Data Mining Software: An Update, *SIGKDD Explor. Newsl.* **11**(1): 10–18.

Ham, H.-S. and Choi, M.-J. [2013]. Analysis of android malware detection performance using machine learning classifiers, *ICT Convergence (ICTC), 2013 International Conference on*, pp. 490–495.

Herman, P. [2009]. Tcpstat. Online: http://www.frenchfries.net/paul/tcpstat/.

Holte, R. [1993]. Very simple classification rules perform well on most commonly used datasets, *Machine Learning* **11**(1): 63–90.
URL: *http://dx.doi.org/10.1023/A%3A1022631118932*

Institute, I. [2016]. Evolution in the World of Cyber Crime, *Technical report*, Infosec Institute.
URL: *http://resources.infosecinstitute.com/evolution-in-the-world-of-cyber-crime/gref*

Internet Security Threat Report Volume 20 [2015]. *Technical report*, Symantec.

Internet Security Threat Report Volume 22 [2017]. *Technical report*, Symantec.

Isohara, T., Takemori, K. and Kubota, A. [2011]. Kernel-based behavior analysis for android malware detection, *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*, IEEE, pp. 1011–1015.

IT Threat Evolution In Q2 2015 [2015]. *Technical report*, Kaspersky Lab.

John, G. and Langley, P. [1995]. Estimating continuous distributions in bayesian classifiers, *In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, pp. 338–345.

Khan, S., Nauman, M., Othman, A. and Musa, S. [2012]. How secure is your smartphone: An analysis of smartphone security mechanisms, *International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, pp. 76–81.

Kim, H., Smith, J. and Shin, K. G. [2008]. Detecting energy-greedy anomalies and mobile malware variants, *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, ACM, New York, NY, USA, pp. 239–252.

Kohavi, R. [1995]. A study of cross-validation and bootstrap for accuracy estimation and model selection, Morgan Kaufmann, pp. 1137–1143.

Labs, M. [2016]. McAfee Labs Threats Report - December 2016, *Technical report*, McAfee Labs.
URL: *https://www.mcafee.com/au/resources/reports/rp-quarterly-threats-dec-2016.pdf*

Le Cessie, S. and Van Houwelingen, J. C. [1992]. Ridge estimators in logistic regression, *Applied statistics* pp. 191–201.

Liu, H. and Yu, L. [2005]. Toward integrating feature selection algorithms for classification and clustering, *IEEE Transactions on Knowledge and Data Engineering* **17**(4): 491–502.

Liu, L., Yan, G., Zhang, X. and Chen, S. [2009]. Virusmeter: Preventing your cellphone from spies, *12th International Symposium on Recent Advances in Intrusion Detection (RAID)*, Springer, pp. 244–264.

Livshits, B. [2013]. Securibench micro.
URL: *https://suif.stanford.edu/ livshits/work/securibench-micro/*

Martinelli, F., Mercaldo, F. and Saracino, A. [2017]. Bridemaid: An hybrid tool for accurate detection of android malware, *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ACM, pp. 899–901.

Martinelli, F., Mercaldo, F., Saracino, A. and Visaggio, C. A. [2016]. I find your behavior disturbing: Static and dynamic app behavioral analysis for detection of android malware, *Privacy, Security and Trust (PST), 2016 14th Annual Conference on*, IEEE, pp. 129–136.

McAfee Labs Threats Report [2015]. *Technical report*, McAfee Labs.

McAfee Labs Threats Report [2017]. *Technical report*, McAfee Labs. Online: https://www.mcafee.com/us/resources/reports/rp-quarterly-threats-jun-2017.pdf.

Mercaldo, F., Nardone, V., Santone, A. and Visaggio, C. A. [2016]. Ransomware steals your phone. formal methods rescue it, *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, Springer, pp. 212–221.

Micro, T. [2015]. 2016 Trend Micro Security Predictions: The Fine Line, *Technical report*, Trend Micro. Online: http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt-the-fine-line.pdf.

Milosevic, J. and Ferrante, A. and Malek, M. [2017]. Time, Accuracy and Consumption Tradeoff for Efficient Malware Detection. Internal Report.

Milosevic, J., Dittrich, A., Ferrante, A. and Malek, M. [2014]. A resource-optimized approach to efficient early detection of mobile malware, *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, IEEE, pp. 333–340.

Milosevic, J., Ferrante, A. and Malek, M. [2015]. Poster: A general practitioner or a specialist for your infected smartphone?
URL: *http://www.ieee-security.org/TC/SP2015/posters/paper*$_1$6.*pdf*

Milosevic, J., Ferrante, A. and Malek, M. [2016a]. Malaware: Effective and efficient run-time mobile malware detector, *The 14th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2016)*, IEEE Computer Society Press, IEEE Computer Society Press, Auckland, New Zealand.

Milosevic, J., Ferrante, A. and Malek, M. [2016b]. Trojans families identification using dynamic features and low complexity classifiers, *EICAR 2016, 24th Annual European Institute for Computer Antivirus Research Conference on Trustworthiness in IT Security Products*, Nuremberg, Germany.

Milosevic, J., Ferrante, A. and Malek, M. [2016c]. What does the memory say? towards the most indicative features for efficient malware detection, *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pp. 759–764.

Milosevic, J., Ferrante, A. and Regazzoni, F. [2015]. Security challenges for hardware designers of mobile systems, *2015 Mobile Systems Technologies Workshop (MST)*.

Milosevic, J., Malek, M. and Ferrante, A. [2016]. A friend or a foe? detecting malware using memory and cpu features, *SECRYPT 2016, 13th International Conference on Security and Cryptography*, SciTePress Digital Library, SciTePress Digital Library, Lisbon, Portugal.

Milosevic, J., Malek, M. and Ferrante, A. [2017]. *Runtime Classification of Mobile Malware for Resource-constrained Devices*, Springer International Publishing AG.

Milosevic, J., Regazzoni, F. and Malek, M. [2017]. *Malware Threats and Solutions for Trustworthy Mobile Systems Design*, Springer.

Moser, A., Kruegel, C. and Kirda, E. [2007]. Limits of static analysis for malware detection, *Twenty-Third Annual Computer Security Applications Conference*, pp. 421–430.

Nair, V. P., Jain, H., Golecha, Y. K., Gaur, M. S. and Laxmi, V. [2010]. Medusa: Metamorphic malware dynamic analysis using signature from api, *Proceedings of the 3rd International Conference on Security of Information and Networks*, SIN '10, ACM, New York, NY, USA, pp. 263–269.
URL: *http://doi.acm.org/10.1145/1854099.1854152*

New Rules: The Evolving Threat Landscape in 2016 [2015]. *Technical report*, FortiGuard Labs.

Oberheide, J., Veeraraghavan, K., Cooke, E., Flinn, J. and Jahanian, F. [2008]. Virtualized in-cloud security services for mobile devices, *1st Workshop on Virtualization in Mobile Computing*, MobiVirt '08, ACM, pp. 31–35.

Percoco, N. J. and Schulte, S. [2012]. Adventures in bouncerland, failures of automated malware detection within mobile application markets, *Technical report*, Trustwave Holdings, Inc. Black Hat Convention.

Platt, J. C. [1999]. Advances in kernel methods, MIT Press, Cambridge, MA, USA, chapter Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pp. 185–208.
URL: *http://dl.acm.org/citation.cfm?id=299094.299105*

Portokalidis, G., Homburg, P., Anagnostakis, K. and Bos, H. [2010]. Paranoid android: versatile protection for smartphones, *26th Annual Computer Security Applications Conference (ACSAC)*, ACM, pp. 347–356.

Quinlan, J. R. [1993]. *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Rastogi, V., Chen, Y. and Jiang, X. [2013]. Droidchameleon: evaluating android anti-malware against transformation attacks, *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, ACM, pp. 329–334.

Reina, A., Fattori, A. and Cavallaro, L. [2013]. A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors, *EuroSec* .

Schmidt, A.-D., Peters, F., Lamour, F. and Albayrak, S. [2007]. Monitoring smartphones for anomaly detection, *1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, MOBILWARE '08, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 40:1–40:6.

Shabtai, A., Fledel, Y. and Elovici, Y. [2010]. Securing Android-Powered Mobile Devices Using SELinux, *IEEE Security and Privacy* **8**: 36–44.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C. and Weiss, Y. [2012]. "andromaly": A behavioral malware detection framework for android devices, *J. Intell. Inf. Syst.* **38**(1): 161–190.
URL: *http://dx.doi.org/10.1007/s10844-010-0148-x*

Shlens, J. [2005]. A tutorial on principal component analysis, *Systems Neurobiology Laboratory, Salk Institute for Biological Studies*.

Smalley, S., Vance, C. and Salamon, W. [2001]. Implementing SELinux as a Linux Security Module, *Technical report*, US National Security Agency. `http://www.nsa.gov/research/_files/publications/implementing_selinux.pdf`.

Song, S., Kim, B. and Lee, S. [2016]. The effective ransomware prevention technique using process monitoring on android platform, *Mobile Information Systems* **2016**.

Spreitzenbarth, M., Echtler, F., Schreck, T., Freling, F. C. and Hoffmann, J. [2013]. Mobilesandbox: Looking deeper into android applications, *28th International ACM Symposium on Applied Computing (SAC)*.

Spreitzenbarth, M., Freiling, F., Echtler, F., Schreck, T. and Hoffmann, J. [2013]. Mobile-sandbox: Having a deeper look into android applications, *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, ACM, New York, NY, USA, pp. 1808–1815.

Tan, D. J., Chua, T.-W., Thing, V. L. et al. [2015]. Securing android: a survey, taxonomy, and challenges, *ACM Computing Surveys (CSUR)* **47**(4): 58.

Truong, H. T. T., Lagerspetz, E., Nurmi, P., Oliner, A. J., Tarkoma, S., Asokan, N. and Bhattacharya, S. [2013]. The company you keep: Mobile malware infection rates and inexpensive risk indicators., *CoRR* .

Viega, J. and Thompson, H. [2012]. The state of embedded-device security (spoiler alert: It's bad), *IEEE Security and Privacy* **10**(5): 68–70.

Vinod, P., Laxmi, V. and Gaur, M. [2012]. Reform: Relevant features for malware analysis, *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, pp. 738–744.

Virus Total [n.d.]. Files Analyser. Online:`https://www.virustotal.com/`.

Wang, X., Jhi, Y.-C., Zhu, S. and Liu, P. [2009]. Detecting software theft via system call based birthmarks, *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, IEEE, pp. 149–158.

Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M. and Wu, K.-P. [2012]. Droidmat: Android malware detection through manifest and api calls tracing, *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*, pp. 62–69.

Xie, L., Zhang, X., Seifert, J.-P. and Zhu, S. [2010]. pbmds: A behavior-based malware detection system for cellphone devices, *Proceedings of the Third ACM Conference on Wireless Network Security*, WiSec '10, ACM, New York, NY, USA, pp. 37–48.

Yang, T., Yang, Y., Qian, K., Lo, D. C.-T., Qian, Y. and Tao, L. [2015]. Automated detection and analysis for android ransomware, *IEEE 17th International Conference on High Performance Computing and Communications, IEEE 7th International Symposium on Cyberspace Safety and Security, IEEE 12th International Conference on Embedded Software and Systems*, IEEE, pp. 1338–1343.

Yang, W., Xiao, X., Andow, B., Li, S., Xie, T. and Enck, W. [2015]. Appcontext: Differentiating malicious and benign mobile app behaviors using context, *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, IEEE Press, Piscataway, NJ, USA, pp. 303–313.
URL: *http://dl.acm.org/citation.cfm?id=2818754.2818793*

Yen, T.-F. and Reiter, M. K. [2008]. Traffic aggregation for malware detection, *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '08, Springer-Verlag, Berlin, Heidelberg, pp. 207–227.
URL: *http://dx.doi.org/10.1007/978-3-540-70542-0*$_1$1

Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M. and Yang, L. [2010]. Accurate online power estimation and automatic battery behavior based power model generation for smartphones, *Proceeding of International Conference on Hardware/Software Codesign and System Synthesis*, pp. 105–114.

Zhou, Y. and Jiang, X. [2012]. Dissecting android malware: Characterization and evolution, *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, IEEE Computer Society, Washington, DC, USA, pp. 95–109.
URL: *http://dx.doi.org/10.1109/SP.2012.16*